# LE LIVRE DU MSX

Daniel MARTIN



BCM

# LE LIVRE DU MSX

PAR D. MARTIN

BCM 1984

AUTRES OUVRAGES EDITES CHEZ B.C.M.

Programmes internes du PET/CBM (3000,4000,8000) par B. MICHEL.

Le livre du VIC par B. MICHEL.

Le livre du 64 par B. MICHEL.

MINIDISQUES MAGNETIQUES DISTRIBUES PAR B.C.M.

Le disque du 64

Le disque du MSX

2ème édition Février 1985


24, route de la Sapinière - 4960 BANNEUX - BELGIQUE
ISBN 2-871110-002-6


Les livres édités par B.C.M. sont distribués par :
- En Belgique, SAPECA s.a., 5, avenue de la Ferme Rose, 1180 UCCLE
- En France, PSI diffusion, BP86, 77402 LAGNY S/MARNE CEDEX
- Au Canada, SCE inc., 65, avenue Hillside Westmount, QUEBEC H3Z1W1

# INTRODUCTION.

Le système MSX est une des machines les plus puissantes que je connaisse dans sa gamme de prix. Il est judicieusement servi par un BASIC très élaboré signé MICROSOFT.

Au niveau des possibilités graphiques, sonores, de gestion des interruptions et d'extentions, peu de micro-ordinateurs bas de gamme peuvent rivaliser avec lui.

J'espère que vous trouverez réponse à toutes vos questions à la lecture de ce livre et que vous l'utiliserez quotidiennement comme manuel de référence tant au point de vue HARDWARE (matériel) que SOFTWARE (logiciel).

Il ne vous reste plus qu'à en faire le meilleur usage pour vous décharger des aléas des techniques spécifiques et concentrer toute votre attention sur la logique de votre application.

Je rappelle que pour tirer un maximum de profit de la lecture de ce livre, des connaissances de base de l'assembleur sont préférables et qu'une bonne connaissance du BASIC classique et de la notation hexadécimale est absolument indispensable.

Le standard MSX, puisqu'il faut bien parler de standard, c'est la garantie de trouver à l'intérieur de l'ordinateur les éléments suivants : un processeur Z80 à 4 Mhz, un VDP 9918, des ports d'entrée/sortie à des adresses bien définies, un interface pour manettes de jeu, et surtout, 32 K de ROM contenant le BASIC MICROSOFT avec un jeu d'instructions bien déterminé.

Ce standard assurera une compatibilité totale entre les logiciels écrits pour l'une ou pour l'autre machine.

Pour terminer, voici une liste non exhaustive des principales compagnies qui ont signé le protocole MSX :

CANON, FUJITSU, GENERAL, HITACHI, JVC, MITSUBISHI, NEC, PIONEER, SANYO, SONY, TOSHIBA, VICTOR, YAMAHA, YASHICA, YENO...

BONNE LECTURE,

D. MARTIN

Novembre 1984.

## L'auteur.

Après quelques mois comme enseignant au Ministère de l'Education Nationale, l'auteur, attiré par la micro-informatique depuis 1978, a travaillé comme COMPUTER MANAGER pour TANDY CORPORATION durant un an et demi. Un bref passage chez APPLE aux Pays-Bas, et, depuis 1981, il est ingénieur système chez INTERTECHNIQUE, le constructeur français spécialisé en mini-ordina--teurs base de données.

Une diskette reprenant tous les programmes du présent ouvrage est disponible au format MSX et vendue au prix de 1800 FB (300 FF) par B.C.M.

Vous pouvez obtenir cette diskette chez B.C.M. s.c. en envoyant la somme par mandat international ou coupon réponse international (pour la Belgique, un mandat postal normal suffit).

N'oubliez pas de spécifier le type de votre disque :

5"1/4 (SANYO - SPECTRA),
3"1/2 (SONY) et
2"8 (QUICK DRIVE DAEWO).

Les chapitres 6 et 7 et l'annexe du présent ouvrage ont été composés au moyen d'un système de traitement de texte Intertechnique (IN 500) associé à une imprimante IN 5712 de type Général Electric en fonte gothique.

Les premiers chapitres ont été dactylographiés sur une imprimante IBM à sphère (LETTER GOTHIC 12).

## TABLE DES MATIERES

# TABLE DES MATIERES



# TABLE DES MATIERES

TABLE DES MATIERES



# 1 ORGANISATION INTERNE DE L'ORDINATEUR MSX

## 1.1 Généralités.

Le système MSX est articulé autour d'un microprocesseur Z80 de chez ZILOG. Ce microprocesseur constitue l'unité centrale du système. L'horloge qui détermine sa vitesse d'exécution est régularisée par un quartz de 3,57 Mhz.

Autour du Z80 on trouve d'abord 32 kilo-octets de mémoires mortes (ROM) contenant le BASIC MICROSOFT étendu.

On trouve ensuite une mémoire vive (RAM) de 8, 16, 32 ou 64 kilo-octets suivant les modèles et les constructeu

Le circuit périphérique principal est certainement le processeur de gestion d'écran (VDP pour Vidéo Display Processor) qui est un circuit spécialisé fabriqué par TEXAS INSTUMENTS sous le numéro TMS-9918. Ce circuit, qui sera analysé en détail dans le chapitre 2, gère l'affichage des textes, des graphiques, des SPRITES (lutins) ainsi que les diverses informations sur les couleurs de fond, de bord et de traçages.

Le TMS-9918 a une mémoire personnelle de 16 kilobytes constituée de 8 circuits 4116. Cette mémoire n'est pas directement en contact avec le BUS du Z80 et nous verrons comment y accéder au chapitre suivant.

Un autre circuit spécialisé est utilisé pour la production des effets sonores : c'est l'AY3-8910qui est un générateur de sons complexes. Il sera étudié en détail dans le chapitre 3. Outre ses possibilités sonores, ce circuit possède 2 ports d'entrée/sortie qui sont utilisés principalement pour la lecture des manettes de jeu (JOYSTICKS) et des palettes analogiques.

Enfin un dernier circuit est utilisé pour la gestion de la cassette, la lecture du clavier et la commutation des mémoires (BANKING). C'est le PPI 8255. Il sera étudié en profondeur dans le chapitre 4.

Le controleur d'imprimante est composé uniquement de circuits logiques élémentaires.

Ajoutez à tout cela la logique de décodage et les buffers (tampons), la partie analogique de mixage des effets sonores, de génération de la composite vidéo, l'alimentation et vous avez une vue schématique de l'intérieur de votre ordinateur MSX.

Un schéma bloc général représentant les différents modules décrits ci-dessus est donné à la fin de ce chapitre.

Pour les heureux possesseurs d'un connecteur d'extension multiple,la configuration de base peut s'étoffer de plusieurs périphériques.

Le contrôleur pour lecteur de disquettes est certaine- -ment le plus important et le plus indispensable des périphériques. Il est livré avec le DOS (SED) original MSX qui rajoute de nombreuses commandes au BASIC de base de la machine.

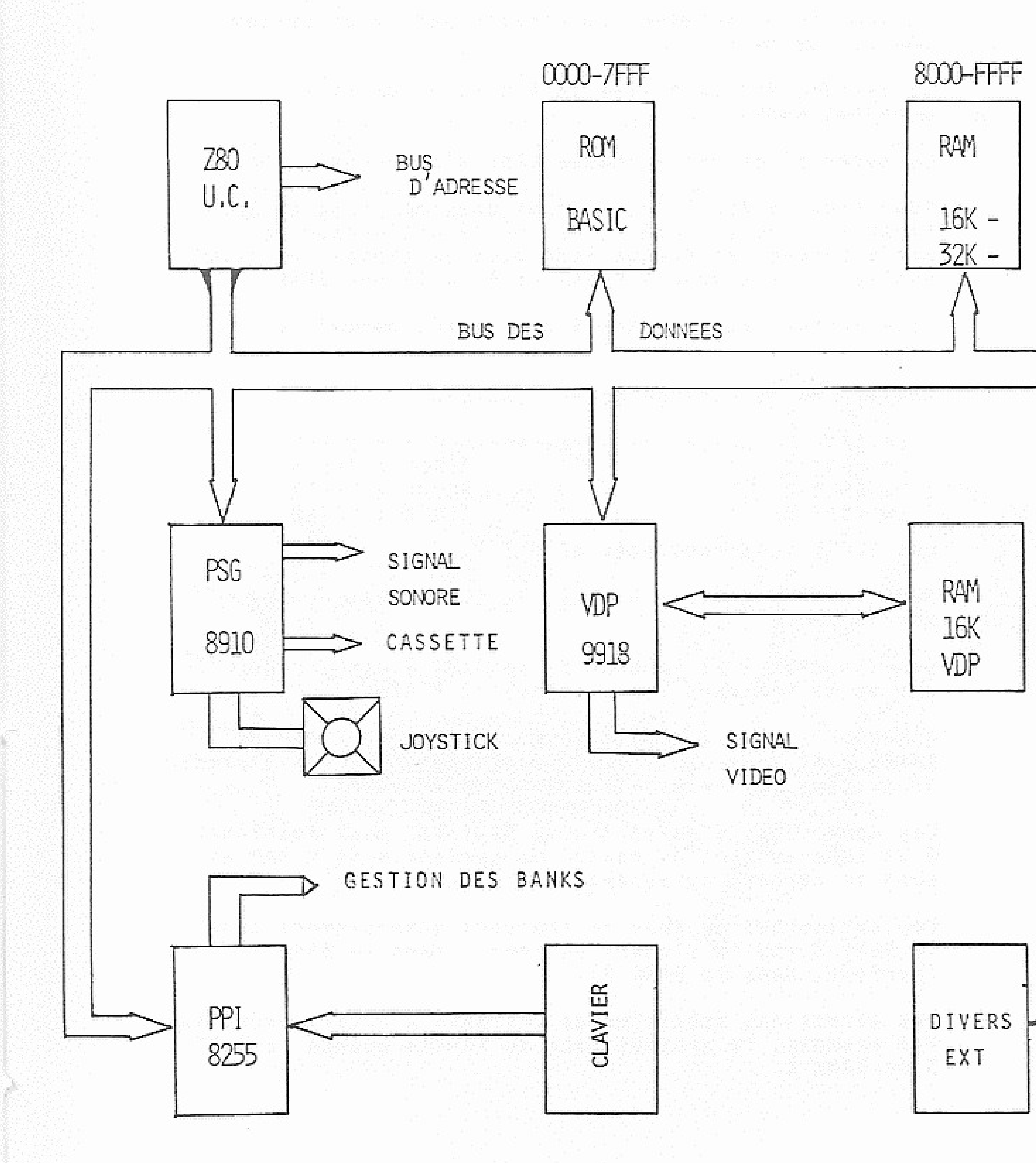
Le connecteur d'extension permet aussi d'ajouter des mémoires supplémentaires au système. La gestion de ces mémoires sera envisagée au chapitre 4 car elle est gérées par un port PPI 8255.

Après toutes ces généralités un peu ennuyeuses, passons aux choses sérieuses. Si vous ne pόssédez pas sur le bout des doigts les instructions PEEK, POKE, USR, OUT et INP ; BASE et VDP, commencez votre lecture par le chapitre 6.

Préparez-vous à entrer dans le monde merveilleux du MSX. Fermez votre porte à clef, décrochez votre télé- -phone, envoyez votre conjoint et vos enfants en vacances et tournez la page pour découvrir les dessous du MSX.



## 1.2 Schéma général de l'unité centrale.



## 1.3 Structure de la mémoire.

La mémoire directement adressable par le processeur Z80 est de 64 K.

Le système MSX la divise en 4 parties de 16 K appelées BANKS.

Le matériel permet à chaque BANK d'adresser un SLOT.

Pour plus de facilité, il faut considérer qu'un SLOT (trad littérale : FENTE) est un sélectionneur de périphérique, et chaque BANK peut en choisir un parmi quatre. Il y a donc 4 BANKS et 4 SLOTS par BANK.

Nous verrons au chapitre 4 que le PPI permet la gestion de ces SLOTS.

Analyse de la configuration standard :

| | | |
|---|---|---|
| Le BANK 0 | occupe les adresses | 0000H à 3FFFH |
| Le BANK 1 | " " " | 4000H à 7FFFH |
| Le BANK 2 | " " " | 8000H à BFFFH |
| Le BANK 3 | " " " | C000H à FFFFH |

Les SLOTS sont numérotés de 0 à 3.

Dans le BANK 0 et le BANK 1, le SLOT 0 est occupé par la ROM.

Dans le BANK 2 et le BANK 3, le SLOT 0 est occupé par de la RAM dans les systèmes 32 K RAM.

REMARQUE : le MSX permet de mettre les 32 K RAM des BANKS 2 et 3 dans n'importe quel SLOT, l'initialisation du système les reconnaissant automatiquement.

Les cartouches d'extension de 32 K RAM pour les BANKS 0 et 1 permettent de porter le système à 64 K RAM et sont en général adressées par le SLOT 1.

Les cartouches de jeux se chargent généralement dans le SLOT 2 et, la plupart du temps, dans le BANK 1 (parfois, dans le BANK 2).

Les extensions spéciales se chargent à divers endroits. Par exemple, le synthétiseur du YAMAHA occupe le SLOT 3 du BANK 1.

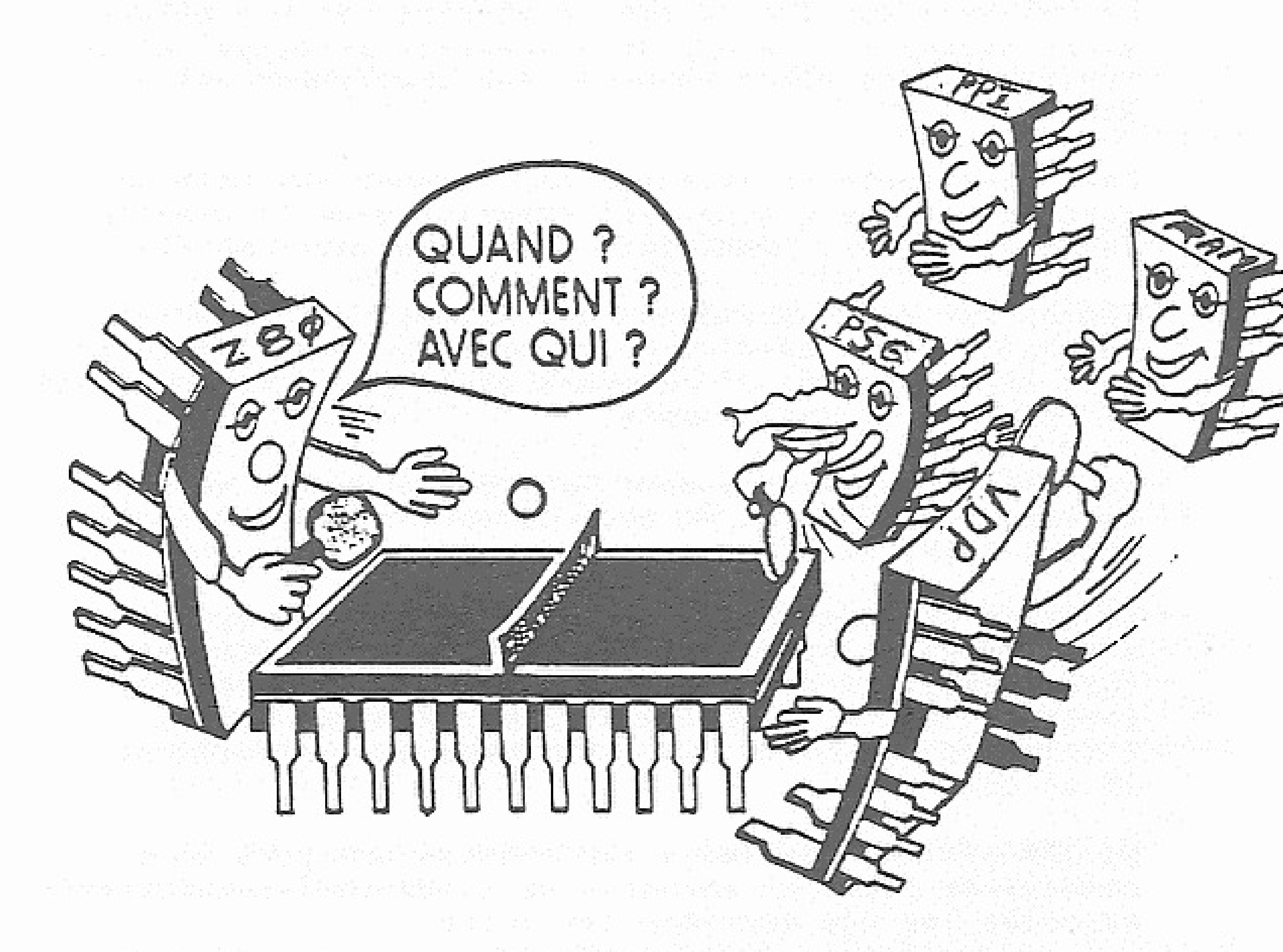


La ROM étant structurée de façon à disposer toutes les fonctions d'entrée/sortie et de gestion des péri-phériques (VDP, PSG, PVI) dans le BANK 0, le BANK 1 ne sert qu'aux instructions BASIC.

C'est donc le BANK 1 qui est remplacé le plus souvent lors du chargement de cartouche en langage machine.

Nous appellerons dès à présent BIOS la partie BANK 0 de la ROM (0000H à 3FFFH), et BASIC la partie BANK 1.

Nous verrons au chapitre 7 quelques exemples de manipulation des SLOTS.



## 2 LE PROCESSEUR DE GESTION D'ECRAN

### 2.1 Généralités

Le processeur de gestion d'écran est basé sur un circuit complexe construit par TEXAS INSTRUMENT, le TMS-9918.

Ce circuit gère tous les signaux de contrôle nécessaires à la génération d'une image vidéo. Il s'occupe aussi du rafraîchissement de sa propre mémoire, du stockage des informations dans ladite mémoire et du rappel de ces informations.

Le TMS-9918 possède sa propre mémoire, elle n'occupe aucun espace sur le BUS du processeur principal et par conséquent, la place réservée aux programmes est plus étendue.

Ceci représente un avantage par rapport aux machines dont la mémoire d'écran est comprise dans la mémoire centrale (ATARI, COMMODORE, TRS80 modèles I et III ...).

Toutes les roses possèdent des épines, voyons l'incon-
-vénient. Si la mémoire n'est pas sur le BUS, elle ne peut être atteinte directement et le jeu d'instructions se réduit de façon notable.

En effet, pour écrire dans la mémoire du VDP encore appelée VIDEORAM, on ne peut se servir que des instruc-
-tions IN(P) et OUT.

Le Basic contient deux instructions qui permettent d'écrire et de lire dans la VIDEORAM comme POKE et PEEK permettent de lire et d'écrire dans la mémoire centrale. Ces 2 instructions sont VPOKE et VPEEK, l'interpréteur BASIC les transforme en instructions IN et OUT.

Le VDP affiche une image sur l'écran qui peut être considérée comme un ensemble de plans semi-transparents disposés les uns derrière les autres.

Les objets qui se trouvent sur les plans les plus proches de la personne qui regarde l'écran ont la plus grande priorité. Dans le cas où deux ou plusieurs objets occupent le même emplacement sur l'écran mais pas le même plan, l'objet ayant la plus haute priorité est visualisé.



Les 32 premiers plans peuvent contenir un et un seul SPRITE ou LUTIN. (Un SPRITE est un objet matricé sur 8X8, 16X16 ou 32X32 points qui est défini par ses coor-
-données verticales et horizontales dans la VIDEO RAM.)

Les points du plan non occupés par le SPRITE sont trans-
-parents. le SPRITE étant défini simplement par ses coordonnées, le déplacer dans l'écran ne présente aucune difficulté comme nous le verrons par la suite.

Derrière les 32 plans réservés aux SPRITES, on trouve le plan réservé aux textes et aux graphiques normaux. Ensuite, on trouve le plan de couleur de fond, ce plan est plus large que les autres, il forme un bord autour des autres plans.

Le dernier plan du VDP n'est pas utilisé sur le système MSX. il est réservé à une entrée vidéo extérieure.

Les 32 plans réservés aux SPRITES sont disponibles en mode graphique ou multicolore, ils ne sont pas utilisa-
-bles en mode texte (mode 0) où ils sont automatiquemen
transparents.

Chacun des SPRITES peut recouvrir une zone de 8X8, 16X1
ou 32X32 points sur le plan. Chaque partie d'un plan non occupée par un SPRITE est transparente. Chaque poin
du SPRITE peut être transparent ou non. Le SPRITE 0 est le SPRITE le plus prioritaire, le SPRITE 31 est le SPRITE le moins prioritaire. Quand un point est trans-
-parent, la  uleur du plan suivant est visualisée ; s'il n'est pas transparent, les couleurs des plans suivants sont remplacées par la couleur du point en question.

Il y a une restriction sur le nombre de SPRITES par ligne horizontale, seuls 4 SPRITES peuvent être actifs en même temps sur une même horizontale, les autres sont automatiquement transparents. Seuls les SPRITES actifs peuvent déclencher le sémaphore de coïncidence (voir plus loin).

Le VDP possède 4 modes de fonctionnement appelés respectivement :

Mode TEXTE    :correspondant à l'instruction SCREEN 0.
M. GRAPHIQUE 1:correspondant à l'instruction SCREEN 1.
M. GRAPHIQUE 2:correspondant à l'instruction SCREEN 2.
M. MULTICOLORE:correspondant à l'instruction SCREEN 3.

Les modes graphiques 1 et 2 permettent une résolution de 256 X 192 points soit 49152 points.

Les 49152 points sont divisés en matrices de 8 X 8 points ; il y a donc 32 X 24 matrices de 8 X 8 points, soit 768 matrices.

En mode graphique 1, les 768 matrices peuvent contenir 256 dessins de caractères différents (nous appelerons un dessin de caractère un PATRON). Chaque patron est défini en deux couleurs seulement.

En mode graphique 2, les 768 matrices peuvent contenir 768 patrons. Chaque ligne de 8 points peut être composée de 2 couleurs, ce qui permet de mettre les 16 couleurs sur une seule matrice de 8 X 8 points.

En mode texte, l'écran est divisé en 40 X 24 positions et les patrons sont de 6 X 8 points. Dans ce mode, les SPRITES n'apparaissent pas sur l'écran et 2 couleurs sont possibles pour l'écran complet, une pour le fond et l'autre pour les caractères.

En mode multicolore, l'écran est divisé en 64 X 48 positions et chaque position est une matrice de 8 X 8 points. Une seule couleur est disponible par matrice.

TMS 9918 TEST PROGRAM

| TYPE | PIN | | | PIN | TYPE |
|---|---|---|---|---|---|
| OUT | RAS | 1 | 40 | XTAL1 | IN |
| OUT | CAS | 2 | 39 | XTAL2 | IN |
| OUT | AD7 | 3 | 38 | CCLK | OUT |
| OUT | AD6 | 4 | 37 | GCLK | OUT |
| OUT | AD5 | 5 | 36 | CVTO | OUT |
| OUT | AD4 | 6 | 35 | XVTO | IN |
| OUT | AD3 | 7 | 34 | RSET SYN | IN |
| OUT | AD2 | 8 | 33 | VCC | PWR |
| OUT | AD1 | 9 | 32 | RD0 | IN |
| OUT | AD0 | 10 | 31 | RD1 | IN |
| OUT | R/W | 11 | 30 | RD2 | IN |
| PWR | VSS | 12 | 29 | RD3 | IN |
| IN | MODE | 13 | 28 | RD4 | IN |
| IN | CSW | 14 | 27 | RD5 | IN |
| IN | CSR | 15 | 26 | RD6 | IN |
| OUT | INT | 16 | 25 | RD7 | IN |
| I/O | CD7 | 17 | 24 | CD0 | I/O |
| I/O | CD6 | 18 | 23 | CD1 | I/O |
| I/O | CD5 | 19 | 22 | CD2 | I/O |
| I/O | CD4 | 20 | 21 | CD3 | I/O |

VDP

BROCHAGE DU VDP



## 2.2 Les tables du VDP.

La VIDEORAM peut être décomposée en une série de tables dont la taille et l'adresse varient en fonction du mode d'affichage.

### 2.2.1 La table des noms des patrons (TNP).

En mode texte, c'est un bloc de 960 mémoires contigües qui représente les 960 positions dans l'écran (40 X 24), chaque mémoire contient le numéro du caractère à afficher à cette position dans l'écran.

La première mémoire contient la valeur du caractère à afficher en haut et à gauche de l'écran, la 40ème contient la valeur du caractère à afficher en haut et à droite de l'écran et bien sûr, la dernière (960) contient la valeur du caractère à afficher en bas et à droite de l'écran.

En mode multicolore et en mode graphique 1 ou 2, la TNP fonctionne similairement au mode texte.

### 2.2.2. La table génératrice des Patrons (TGP).

En mode texte, cette table contient le dessin des 256 caractères affichables. Elle est chargée en standard ave les caractères alphanumériques et semi-graphiques que vous connaissez. Ces caractères peuvent être modifiés par programme.

Chaque caractère est défini par une matrice de 6 X 8 points programmée sur 8 octets.

La matrice constituée de 6 points en horizontal sur 8 points en vertical est structurée comme suit : chaque ligne horizontale est programmée sur un octet dont seuls les 6 bits les plus significatifs sont utilisés, le bit le plus significatif définit le point le plus à gauche de l'horizontale.

La TGP permet de définir 256 caractères codés sur 8 octets. En mode texte, la taille de cette table est de 2048 octets.

En mode multicolore, la TGP contient l'information couleur pour chaque patron.

En mode graphique 1, la TGP définit l'état de chaque point dans une matrice de 8 X 8 points et 256 matrices peuvent être définies.

En mode graphique 2, la TGP est une table de 768 matrices de 8 X 8 points (6144 octets).

## 2.2.3 La Table des Couleurs (TC).

Elle est utilisée par les modes graphiques 1 et 2.

En mode graphique 1, la TC définit la couleur de chaque groupe de 8 patrons. Un octet est réservé par groupe. 32 octets sont donc nécessaires.

Chaque octet est divisé en 2 X 4 bits. les 4 bits les plus significatifs (les plus à gauche) définissent la couleur des bits à 1 dans la TGP. Les 4 bits les moins significatifs (les plus à droite), définissent la couleur des bits à 0 dans la TGP.

4 bits permettant le codage de 16 couleurs, le compte y est.

Voici l'équivalence entre les bits et la couleur :

| | |
|---|---|
| 0000 (00) - TRANSPARENT | 1000 (08) - ROUGE MOYEN |
| 0001 (01) - NOIR | 1001 (09) - ROUGE CLAIR |
| 0010 (02) - VERT MOYEN | 1010 (10) - JAUNE FONCE |
| 0011 (03) - VERT CLAIR | 1011 (11) - JAUNE CLAIR |
| 0100 (04) - BLEU FONCE | 1100 (12) - VERT FONCE |
| 0101 (05) - BLEU CLAIR | 1101 (13) - MAGENTA |
| 0110 (06) - ROUGE FONCE | 1110 (14) - GRIS |
| 0111 (07) - CYAN | 1111 (15) - BLANC |

En mode graphique 2, la TC occupe 6144 octets. Elle permet alors de définir deux couleurs pour chaque octet de la TGP, c'est à dire deux couleurs pour chaque série de 8 points horizontaux.

## 2.2.4 La Table Génératrice des Sprites (TGS).

Cette table contient le dessin du SPRITE, le SPRITE est défini sur 8 X 8 ou 16 X 16 bits, cette valeur étant définie dans le registre 1.

Les bits à 1 sont affichés dans la couleur du SPRITE, les bits à 0 sont toujours transparents.



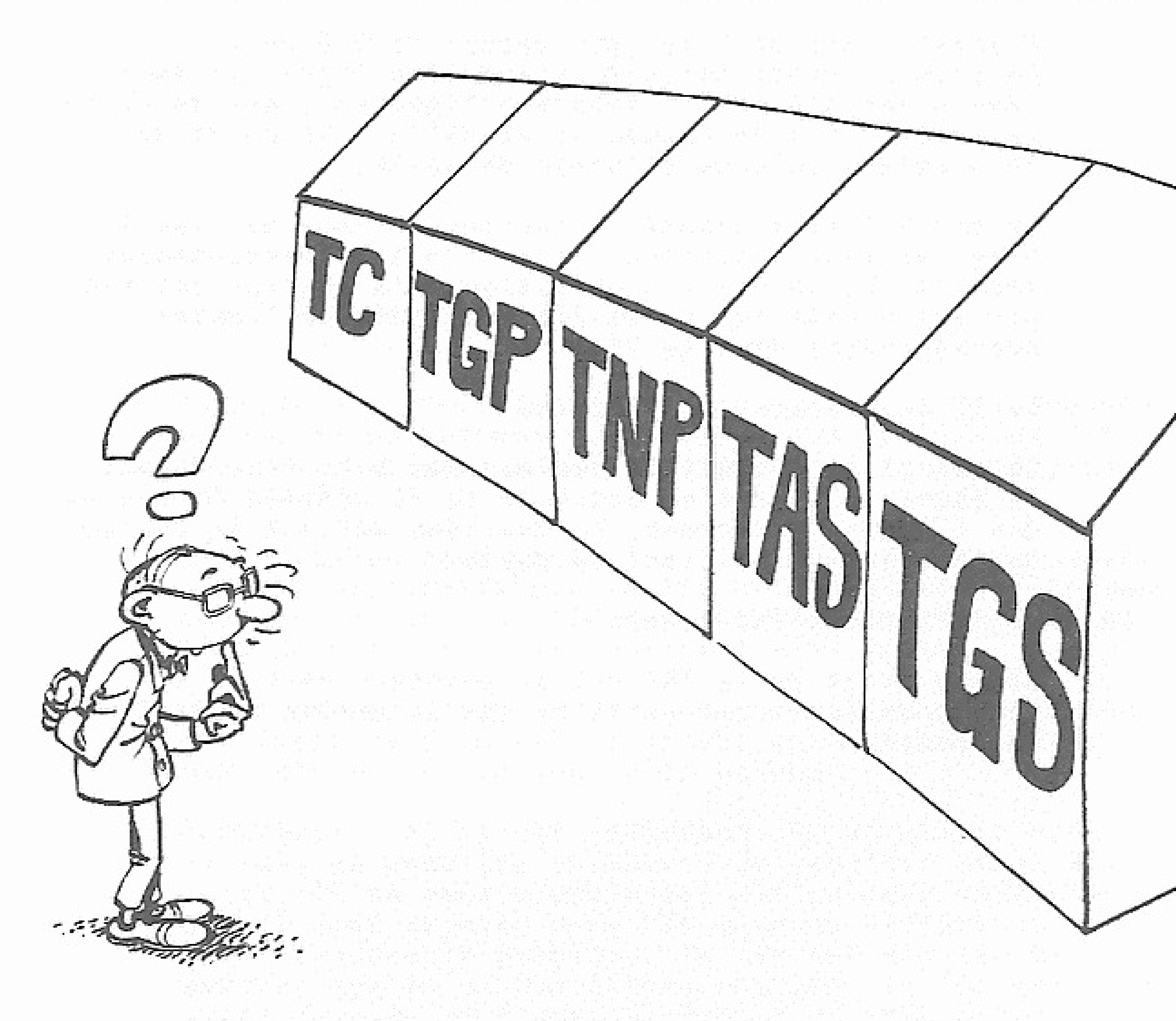
## 2.2.5 La Table des Attributs des Sprites (TAS).

Cette table contient 4 valeurs pour chaque SPRITE, la première valeur est la position verticale du SPRITE sur un octet, la seconde valeur est la position horizontale du SPRITE sur un octet, la troisième valeur est un pointeur vers la TGS qui définit le dessin du SPRITE, la quatrième valeur définit enfin la couleur du SPRITE.

En modifiant les deux premières valeurs de la TAS, vous pouvez donc déplacer un SPRITE sur l'écran.

Comme on a 32 SPRITES composés de 4 octets, il suffit de 128 octets pour définir la TAS.

Des informations supplémentaires sur la TGS et sur la TAS vous seront fournies lors de l'étude des sprites.



## 2.3 Mécanisme d'adressage des différents modes.

### 2.3.1 Mode graphique 1.

Quand on est dans le mode graphique 1,l'écran peut être considéré comme une grille de 32 colonnes et de 24 lignes.

Chaque élément de la grille est un patron de 8 X 8 points.

Trois tables sont utilisées : la TGP, la TNP et la TC.

La TGP contient une librairie des patrons qui peuvent être affichés, elle occupe 2048 octets et peut contenir 256 patrons différents.

8 octets sont utilisés par patron (8 X 8 points). Le premier octet définit la première ligne, le deuxi--ème octet définit la deuxième ligne et ainsi de suite. Le premier bit de chaque octet définit le point de la première colonne et ainsi de suite.

Un bit à 1 sélectionne la couleur définie par les 4 bits les plus significatifs de l'octet correspondant dans la TC, un bit à 0 sélectionne la couleur définie par les 4 bits les moins significatifs de l'octet correspondant dans la TC.

La TC se compose de 32 octets.

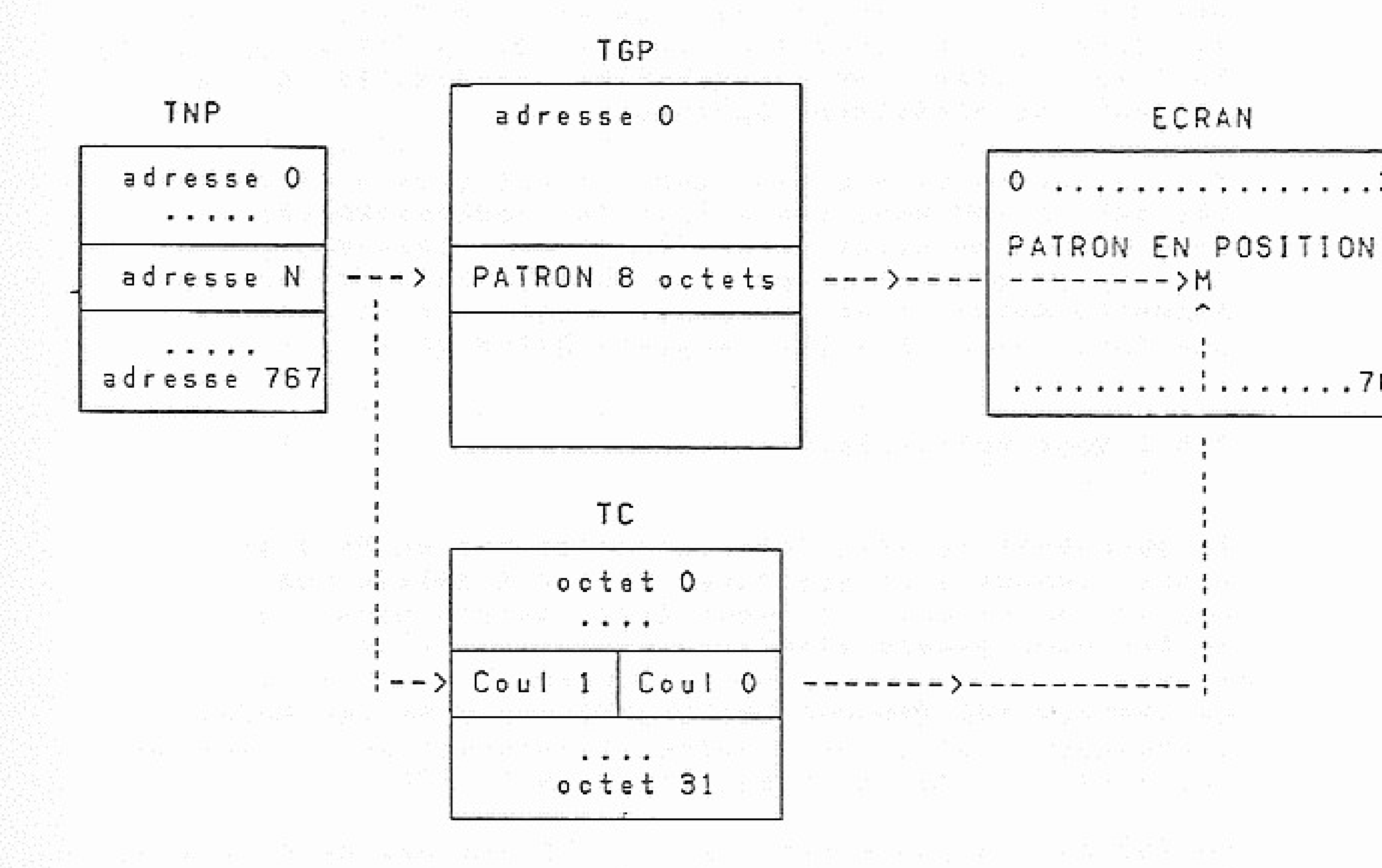
Chaque octet définit la couleur des bits 1 ou 0 dans le patron. Le premier octet de la TC définit la couleur des 8 premiers patrons, le deuxième définit la couleur des 8 suivants et ainsi de suite.

La TNP occupe 768 octets.

Chaque octet de la TNP est un pointeur vers la TGP. Il y a un octet par position sur la grille de 32 X 24 patrons.



Shéma d'adressage :




### 2.3.2 Mode graphique 2.

Ce mode est similaire au mode graphique 1 à l'exceptio du nombre de patrons qui passe à 768 au lieu de 256, e des possibilités de couleurs qui sont beaucoup plus étendues.

Les 3 tables sont utilisées, la TNP contient toujours 768 entrées possibles, la TGP est agrandie 3 fois pour pouvoir stocker les 768 patrons et occupe maintenant 6144 octets, la TC est portée à 6144 octets.

La TC permet ainsi de stocker une information couleur par ligne de 8 points. 8 octets sont donc utilisés pour définir la couleur d'un patron.

REMARQUES : A) La TNP contenant 768 pointeurs vers la TGP, un problème se pose : le pointeur étant d'un octet, il ne peut prendre que 256 valeurs possibles. Or, il faut pointer vers 768 patrons différents. Pour résoudre le problème, la TNP est divisée en 3 parties égales de 256 octets chacune, la TGP est elle aussi divisée en 3 parties égales de 2048 octets chacune. Chaque partie de la TNP est en relation biunivoque avec une partie de la TGP.

D'une manière générale, on peut considérer que l'écran est divisé en trois parties égales : le tiers supérieur est géré par les premières parties de la TNP et de la TGP, le tiers médian, par les deuxièmes parties et le tiers bas par les troisièmes parties.

B) D'un point de vue pratique, il est plus aisé de charger la TNP avec trois fois les nombres consécutifs de 0 à 255. De cette façon, la TNP n'intervient pas dans la programmation et vous pouvez considérer la TGP comme la mesure d'écran. C'est d'ailleurs de cette façon que fonctionne le BASIC en mode SCREEN 2.

### 2.3.3 Mode multicolore.

En mode multicolore, l'écran est divisé en 64 X 64 blocs. Chaque bloc est formé de 4 X 4 points. La couleur de chaque bloc peut être choisie parmi les 15 couleurs disponibles.

La TNP est identique à celle utilisée dans les modes graphiques 1 et 2, par contre, l'information couleur ne provient plus de la TC mais bien de la TGP.

La TGP est toujours définie sur 768 entrées de 8 octets, mais seuls 2 octets de chaque entrée sont utilisés.

Les 2 octets spécifient 4 couleurs, chaque couleur portant sur un bloc de 4 X 4 points dans une matrice de 8 X 8 points.

Les 4 bits les plus significatifs du premier octet définissent la couleur du bloc supérieur gauche. Le deuxième octet définit la couleur des deux blocs inférieurs.

La localisation des deux octets parmi le groupe de 8 de chaque entrée de la TGP dépend de la position du bloc sur l'écran.

Ainsi, la couleur des 32 premiers blocs est définie par les deux premiers octets de chaque groupe de 8 octets de la TGP. La couleur des 32 blocs suivants est définie par les octets 3 et 4 du groupe d'octets de la TGP. Les 32 suivants par les octets 5 et 6, les 32 suivants par les octets 7 et 8, et ensuite, on recommence avec les deux premiers.

En résumé, dans le mode multicolore, 2 tables sont utilisées, la TGP et la TNP. La TNP comporte 768 octets et la TGP 1536 octets (24 X 32 X 8).



### 2.3.4 Mode texte.

En mode texte, l'ecran est donc divisé en 40 X 24 positions, chaque position permettant l'affichage d'une matrice de 6 X 8 points.

Le mode texte n'utilise que la TGP et la TNP.

La TGP permet la définition de 256 patrons ou caractères, chaque caractère étant défini dans une matrice de 8 X 8 bits ou 8 octets.

La TNP, comme à son habitude, pointe sur le caractère défini dans la TGP et est en correspondance octet par octet avec la position du caractère sur l'écran.

La TNP se compose donc de 960 octets et la TGP, de 2048 octets.

Une seule couleur est disponible pour tout l'écran, elle est définie au niveau du registre 7 comme nous allons le voir lors de l'étude des registres.

## 2.4 Les registre du VDP.

Les 9 registres du VDP définissent les différents paramètres du VDP ainsi que les adresses de base pour les différentes tables qui se trouvent dans la VIDEORAM.

Rappel : la VIDEORAM est composée de 8 mémoires de 16K X 1 bit permettant de mémoriser 16384 positions numé-rotées en hexadécimal de 0000 à 3FFF.

Analysons les 9 registres du VDP :

### 2.4.1 Le registre 0.

Seul le bit 1 (les bits étant numérotés de 0 à 7 de droite à gauche) nous intéresse, il contient 1 si on est en mode graphique 2, et 0 dans tous les autres cas.

Tous les autres bits doivent être à l'état 0.

### 2.4.2 Le registre 1.

Le bit 7 est toujours à 1 (RAM 4116).
Le bit 6 est 0 si l'image ne doit pas être affichée et 1 si l'image doit être active. Si l'image n'est pas activée, seule la couleur du bord est affichée.
Le bit 5 est utilisé pour les interruptions, 0 = inter-ruptions interdites, 1 = interruptions autorisées.
Le bit 4 est 1 si on est en mode texte, et 0 dans les autres cas.
Le bit 3 est 1 si on est en mode multicolore et 0 dans les autres cas.
Le bit 2 est toujours 0.
Le bit 1 indique la taille des SPRITES, 0 pour 8 X 8 et 1 pour les 16 X 16 points.
Le bit 0 indique le facteur d'agrandissement des SPRITES, 0 indique la taille normale et 1 indique une multipli-cation par 2 soit alors 16 X 16 ou 32 X 32 points.

### 2.4.3 Le registre 2.

Il contient les 4 bits les plus significatifs de l'adresse de la TNP n'importe où dans la VIDEORAM par pas de 1K car chaque adresse de table se trouve codée sur 14 bits (16K de VIDEORAM).
16 adresses sont donc possibles et l'adresse de la TNP vaut le contenu du registre 2 multiplié par 400H ou 1024 en décimal.



Exemple : si le registre 2 contient 0, la TNP commence au début de la VIDEORAM soit à l'adresse 0000 en hexa, si le registre contient 5 la TNP commence à l'adresse 1400 en hexa (début du 6ème K).

### 2.4.4 Le registre 3.

Il contient les 8 bits les plus significatifs de l'adresse de la TC, ce qui permet de disposer la TC n'importe où dans la VIDEORAM par pas de 64 octets (256 positions possibles).

L'adresse de la TC vaut le contenu du registre 3 multiplié par 40H ou 64 en décimal.

En mode graphique 2, la TC étant de 6144 octets, seul le bit le plus significatif est actif (B7). S'il vaut 0, la table commence en 0000H et finit en 17FFH. S'il vaut 1, la table commence en 2000H et finit en 37FFH.
Les autres bits doivent être à 1.
En résumé, en mode graphique 2, le registre 3 peut contenir 127 ou 255.

### 2.4.5 Le registre 4.

Il contient les 3 bits les plus significatifs de l'adresse de la TGP, ce qui permet de disposer la TGP n'importe où dans la VIDEORAM par pas de 2K (8 positions possibles).

L'adresse de la TGP vaut le contenu du registre 4 multiplié par 800H ou 2048 en décimal.

En mode graphique 2, la TGP étant de 6144 octets, seul le bit le plus significatif parmi les 3 bits (B2) est actif. S'il vaut 0, la table commence en 0000H et finit en 17FFH. S'il vaut 1, la table commence en 2000H et finit en 37FFH. Les deux autres bits doivent alors être à 1.
En résumé, en mode graphique 2, le registre R4 peut contenir 3 ou 7.

2.4.6 Le registre 5.

Il contient les 7 bits les plus significatifs de l'adresse de la TAS, ce qui permet de disposer la TAS n'importe où dans la VIDEORAM par pas de 128 octets (128 positions possibles).

L'adresse de la TAS vaut le contenu du registre 5 multiplié par 80H ou 128 en décimal.

2.4.7 Le registre 6.

Il contient les 3 bits les plus significatifs de l'adresse de la TGS, ce qui permet de disposer la TGS n'importe où dans la VIDEORAM par pas de 2K (8 positions possibles.).

L'adresse de la TGS vaut le contenu du registre 6 multiplié par 800H ou 2048 en décimal.

2.4.8 Le registre 7.

Les 4 bits les plus significatifs définissent la couleur des 1 dans la TGP quand on est en mode texte.

Les 4 bits les moins significatifs définissent la couleur des 0 dans la TGP en mode texte, et si on n'est pas en mode texte, ils définissent la couleur du bord.

2.4.9 Le registre d'état.

Ce registre est à lecture seule et donne des informations sur les SPRITES et les interruptions.

Le registre d'état peut être lu à tout moment.
Sa lecture repositionne le sémaphore d'interruption à 0. Cependant, des lecture désordonnées peuvent altérer l'état du sémaphore d'interruption, c'est pourquoi il vaut mieux lire ce registre quand les interruptions sont suspendues.



Analysons la signification des bits de ce registre.

Bit 7 : c'est le sémaphore d'interruption, il est mis à 1 à la fin du SCANNING de la dernière ligne et remis à 0 par une lecture du présent registre.

Bit 6 : c'est le sémaphore de présence de 5 SPRITES ou plus sur la même ligne horizontale, il est mis à 1 quand la présence est détectée et remis à 0 par une lecture du présent registre.

Bit 5 : c'est le sémaphore de coïncidence, il est mis à 1 si 2 ou plusieurs SPRITES ont au moins un point commun et est remis à 0 par une lecture du présent registre.

Bits 4 à 0 : ils contiennent le numéro du cinquième SPRITE qui a causé la montée à 1 du sémaphore de présence.

Remarque : un tableau récapitulatif de l'utilisation des registres est donné dans l'annexe B du présent volume.

2.4.10 Utilisation des registres.

En écrivant dans les registres, vous pouvez déterminer le mode de travail, les localisations des tables, les couleurs des textes, la position des points, des sprites, leurs couleurs....

En lisant le registre d'état, vous pouvez déterminer si 5 SPRITES sont sur la même horizontale ou si 2 SPRITES ou plus sont sur la même position à l'écran.

## 2.5 Ecriture et lecture des registres et de la VIDEORAM.

### 2.5.1 Généralités.

Pour gérer le VDP en BASIC nous avons à notre disposition une instruction/fonction d'écriture/lecture dans les registres (l'instruction/fonction VDP).

Pour son utilisation, référez-vous à la section 6.10.2 du présent volume.

Pour lire et écrire dans la VIDEORAM, nous avons à notre disposition la fonction VPEEK et l'instruction VPOKE. Pour leur utilisation, référez-vous à la section 6.4.2 du présent volume.

En assembleur, il en va autrement. Nous avons à notre disposition des portes d'entrée/sortie utilisables avec IN et OUT uniquement. La partie BIOS de la ROM contient déjà de telles routines.

Evidemment les instructions IN et OUT du BASIC peuvent remplacer VPEEK, VPOKE et VDP.

### 2.5.2. Adresses des ports.

Le port 98H (152) permet d'écrire une valeur dans un registre ou dans la VIDEORAM.

Le port 99H permet de commander la fonction à exécuter ou de fournir l'ordre d'écriture (STATUS COMMAND REGISTER)

### 2.5.3 Ecriture dans la VIDEORAM.

Soit la donnée D (comprise entre 0 et 255) à écrire à l'adresse A (comprise entre 0 et 16383=3FFFH).

a- décomposer l'adresse en 2 parties (1 haute et 1 basse)
b- écrire la partie basse sur le port 99H
c- écrire la partie haute de l'adresse augmentée de 64 (40H) sur le même port (99H).
d- écrire la donnée sur le port 98H.



EXEMPLE : écrire la donnée décimale 200 à l'adresse 1234H (4660 DECIMAL)

Pour plus de facilité, la fonction OUT est écrite comme en BASIC.

a- décomposition de 1234H :
partie haute 12H (18)
partie basse 34H (52)

b- écriture de la partie basse :

OUT &H99,52

c- écriture de la partie haute :

OUT &H99,18+64

d- écriture de la donnée :

OUT &H98,200

En assembleur, grâce aux routines BIOS de la ROM, pour lire le contenu d'une adresse de la VIDEORAM, il suffit de : 1- Charger HL avec la valeur de l'adresse,
2- Charger A avec la valeur à écrire,
3- faire un CALL en 07CDH.

### 2.5.4 Lecture de la VIDEORAM.

Les points a et b sont identiques aux points a et b de la section 2.5.3.

c- écrire la partie haute de l'adresse sur le port 99H (ne pas augmenter la valeur de 64).

d- lire le contenu du port 98H.

EXEMPLE : lire le contenu de l'adresse &H1234

a- décomposer 18 et 52

b- OUT &H99,52

c- OUT &H99,18

d- V=INP(&H98)

En assembleur il suffit de :
1- charger HL avec le contenu de la VIDEORAM
2- faire un CALL en 07D7H
3- on récupère la valeur dans A.

REMARQUES TRES IMPORTANTES :

A) lorsque vous lisez ou écrivez dans la VIDEORAM, l'adresse est auto-incrémentée et une nouvelle lecture ou écriture vous donnera le contenu de la mémoire suivante sans que vous ne deviez fournir une nouvelle adresse au VDP.

L'accès séquentiel de la VIDEORAM, une fois l'adresse de départ fournie, ne demande qu'une seule instruction assembleur par octet à lire ou à écrire.

B) Après que l'adresse de la VIDEORAM ait été fournie au VDP, il faut quelques microsecondes avant que la donnée à écrire ou à lire soit présentée sur le PORT adéquat, aussi, lors de la programmation en assembleur, vous devez faire suivre vos instructions de chargement de l'adresse de 2 instructions EX (SP),HL permettant un petit délai.

2.5.5 Ecriture dans un registre.

Soit la donnée D comprise entre 0 et 255 à écrire dans le registre R compris entre 0 et 7.

Il suffit d'écrire la donnée D sur le port 99H et ensuite d'écrire la valeur R augmentée de 128H sur le même port.

En assembleur, une routine existe. Il suffit de charger B avec la valeur à écrire, C avec le numéro de registre ; et de faire un CALL en 0047H.

2.5.6 Lecture des registres.

En réalité, seul le registre 8 (STATUS REGISTER) peut être lu, les autres registres ne sont pas accessibles au niveau du VDP.

La fonction BASIC VDP ne réalise que la lecture de la région de communication et non une lecture réelle.

Pour lire le registre d'état, il suffit de lire le port 99H. En assembleur un CALL 13EH réalise la lecture du registre d'état, le résultat se trouvant dans A.



2.6 <u>Adresse de base des tables en BASIC.</u>

Il est évident que chaque table possède en standard une adresse de base déterminée par le système.

Analysons la structure de ces tables et leurs adresses pour les 4 modes courants du système, autrement dit le mode texte, le mode graphique 1, le mode graphique 2, et le mode multicolore correspondant aux instructions SCREEN 0, SCREEN 1, SCREEN 2, et SCREEN 3.

2.6.1 Adressage en mode texte.

En mode SCREEN 0 (TEXTE), les adresses comprises entre 0 et 959 de la VIDEORAM (en hexadécimal 0000 à 03BF) contiennent les codes des caractères affichés en position correspondante sur l'écran. Autrement dit, la TNP est positionnée en 0 ou encore, le registre 2 contient 0.

La TGP quant à elle est positionnée en 2048 (en hexa 0800) ou encore, le registre 4 contient 1, ce qui veut dire que les adresses comprises entre 2048 et 4095 (en hexa 0800 et 0FFF) sont utilisées pour le dessin des caractères, celui correspondant à la valeur 0 se trouvant codé sur les octets compris entre 2048 et 2055 (8 octets).

La TNP et la TGP sont seules utilisées en mode TEXTE.

Exemple : nous voulons afficher un caractère dont voici le dessin (1 représente un point allumé et 0, un point éteint et ce au milieu de l'écran),et ce caractère doit remplacer la lettre A.

DESSIN (matrice de 6 X 8)

```
11111100    xxxxxx
10110100    x xx x
10110100    x xx x
11111100    xxxxxx
11111100    xxxxxx
10110100    x xx x
10110100    x xx x
11111100    xxxxxx
```

On calcule la valeur des 8 octets (1 par ligne).

```
ligne 1    11111100 = FCH = 252
ligne 2    10110100 = B4H = 180
ligne 3    10110100 = B4H = 180
ligne 4    11111100 = FCH = 252
ligne 5    11111100 = FCH = 252
ligne 6    10110100 = B4H = 180
ligne 7    10110100 = B4H = 180
ligne 8    11111100 = FCH = 252
```

On détermine la valeur de la lettre A en prenant son code ASCII. A=41H=65.

On détermine l'adresse du milieu d'écran (environ 460).

On écrit la valeur du code de A au milieu de l'écran.

```
10 CLS
20 VPOKE 460,65
```

On détermine la position de A dans la TGP (facile, c'est 2048 + 8 X 65).

Soit AD, cette adresse, il ne reste plus qu'à inscrire les valeurs de chaque ligne dans les 8 adresses successives.

```
30 AD = 2568
40 VPOKE AD,252
50 VPOKE AD+1,180
60 VPOKE AD+2,180
70 VPOKE AD+3,252
80 VPOKE AD+4,252
90 VPOKE AD+5,180
100 VPOKE AD+6,180
110 VPOKE AD+7,252
```

Et le tour est joué!

REMARQUE : bien sûr, les instructions vues ci-dessus doivent se dérouler dans un programme de façon successive. A partir de cet instant, la lettre A de votre clavier est remplacée par le petit dessin.

Listez le programme et regardez le A de AD !



2.6.2 Adressage en mode graphique 1.
mode texte 32 colonnes.

Dans ce mode toutes les tables sont actives.

La TGP qui contient les dessins des 256 caractères en format 8 X 8 est localisée de 0 à 6143 (0000H à 17FFH) autrement dit, le registre 4 contient 0.

La TC commence en 8192 (2000H) et occupe 32 octets, autrement dit, R3 contient 128.

La TNP est positionnée à l'adresse 1800H, R2 vaut donc 6.

La TGS est positionnée à l'adresse 3800H, R6 vaut donc 7.

La TAS est positionnée à l'adresse 1B00H, R5 vaut donc 54

SYNTHESE.

| NOM | ADR. DEBUT | REGISTRE | CONTENU DU REGISTRE. |
|---|---|---|---|
| TGP | 0000H | R4 | 0 |
| TNP | 1800H | R2 | 6 |
| TAS | 1B00H | R5 | 54 ou 36H |
| TC | 2000H | R3 | 128 ou 80H |
| TGS | 3800H | R6 | 7 |

2.6.3 Adressage en mode graphique 2.

Dans ce mode (SCREEN 2), toutes les tables sont actives.

La TGP qui contient l'information sur les points allumés ou éteints se trouve de 0 à 6143 (en hexa, de 0000 à 17FF), autrement dit, le registre 4 contient 3 (voir remarque lors de la description du registre 4).

L'octet 0 contient l'information sur les points de coordonnées 0,0 à 7,0 (format colonne, ligne).

L'octet 1 contient l'information sur les points de coordonnées 0,1 à7,1.

L'octet 7 contient l'information sur les points de coordonnées 0,7 à 7,7.

L'octet 8 contient l'information sur les points de coordonnées 8,0 à 15,0.

Et ainsi de suite.

Pour allumer le point de coordonnée X,Y avec 0⩽X⩽255 et 0⩽Y⩽191 il suffit de faire :

```
VPOKE AD,(2 (7-(XMOD8))) OR VPEEK AD
```

avec AD = X - X MOD 8 + Y MOD 8 + 256 * (Y \ 8)

La TC commence en 8192 et se termine en 14335 (en hexa 2000 à37FF), autrement dit, le registre 3 contient 255 (voir remarque dans la description de R3).

La mémoire est organisée comme suit : chaque adresse contient la couleur d'un groupe de 8 points horizontaux.

L'octet d'adresse 8192 s'occupe des 8 points de coordon--nées 0,0 à 0,7 , le suivant (8193) s'occupe des 8 points de coordonnées 8,0 à 15,0...

Exemple : pour afficher le point de coordonnées X,Y de la couleur CL avec les points éteints de couleur CF :

```
VPOKE AD,V
```

avec AD = Y + 320 + X \ 8 + 8192 et V = 16 * CL + CF.

La TNP est positionnée à partir de l'adresse 1800H et occupe 768 octets. Le registre 2 contient donc 6.

La TGS est positionnée en 3800H. Le registre 6 contient donc 7.

La TAS est positionnée en 1B00H. Le registre 5 contient 36H (54).

SYNTHESE.

| Nom | Adresse début | Adresse fin | Registre | Contenu du registre. |
|---|---|---|---|---|
| TGP | 0000H | 17FFH | R4 | 03H |
| TNP | 1800H | 1AFFH | R2 | 06H |
| TAS | 1B00H | 1FFFH | R5 | 36H |
| TC | 2000H | 37FFH | R3 | FFH |
| TGS | 3800H | 3BFFH | R6 | 07H |

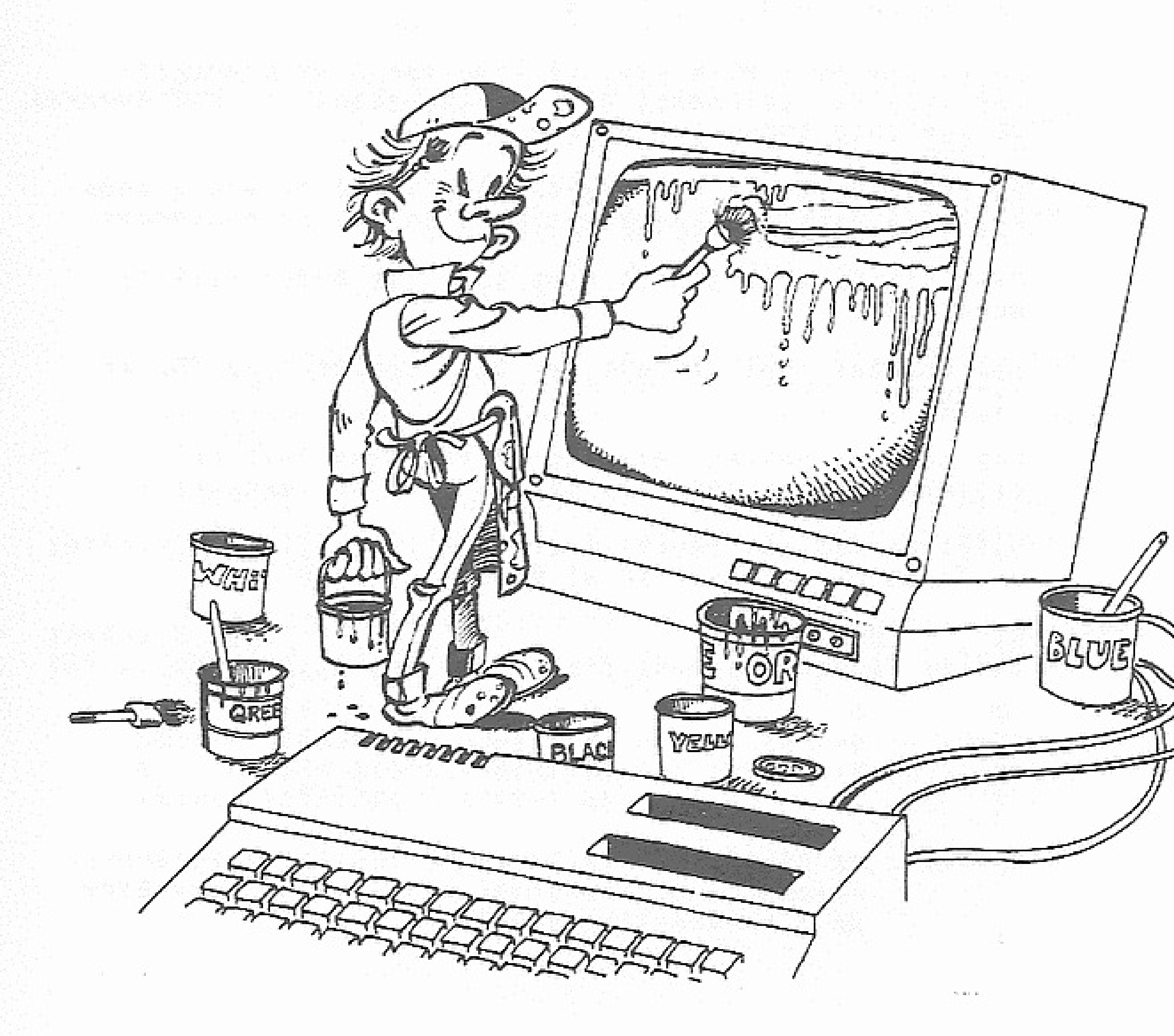


2.6.4 Adressage en mode multicolore.

En mode multicolore, les adresses de la TGS et de la TAS sont identiques à celles du mode graphique 2. La TC n'est pas utilisée.

La TGP commence en 0000H, R4 contient donc 0.

La TNP commence en 0800H, R2 contient donc 2.



## 2.7 LES SPRITES.

### 2.7.1 Généralités.

Le VDP peut afficher 32 sprites sur ses 32 plans les plus prioritaires à raison d'un sprite par plan.

La structure des tables qui définissent les sprites facilite l'animation de ces derniers.

La localisation d'un sprite sur l'écran est définie par le point supérieur gauche de son patron. Ainsi, un sprite de 8 X 8 points affiché en 128,96 occupera les points de 128,96 à 137,104.

Le sprite peut être déplacé simplement en changeant son origine, autrement dit, en changeant les coordonnées de son coin supérieur gauche.

Ce système permet une grande simplicité et une grande rapidité dans la programmation d'objets en mouvement.

Les sprites sont actifs dans tous les modes sauf le mode texte.

Les sprites sont définis par deux tables : la TAS et la TGS.

Les sprites peuvent être de différentes tailles, celles-ci étant définies parles bits du registre 1.

Voici un tableau récapitulatif de la taille des sprites en fonction des bits B0 et B1 du registre 1 :

| B1 taille | B0 agrandissement | dimension | résolution | Nb d'octets dans la TGS |
|---|---|---|---|---|
| 0 | 0 | 8 X 8 | 1 POINT | 8 |
| 1 | 0 | 16 X 16 | 1 POINT | 32 |
| 0 | 1 | 16 X 16 | 2X2 PTS | 8 |
| 1 | 1 | 32 X 32 | 2X2 PTS | 32 |

RAPPEL : en BASIC, la taille et le facteur d'agrandis-sement sont commandés par le 2ème paramètre de l'instruction SCREEN.



SCREEN 2,0 : mode graphique 2, taille 8 X 8.
SCREEN 2,1 : idem 2,0 mais avec agrandissement.
SCREEN 2,2 : mode graphique 2, taille 16 X 16.
SCREEN 2,3 : idem 2,2 mais avec agrandissement.

### 2.7.2 La TAS. en Screen 2 debute à 1B00

La TAS spécifie les coordonnées et la couleur du sprite.

La TAS comporte 4 octets par sprite.

Octet 1 : position verticale du point supérieur gauche.

Octet 2 : position horizontale du coin supérieur gauche.

Octet 3 : pointeur relatif à la TGS.

Octet 4 : Les 4 bits les moins significatifs indiquent la couleur des bits à 1 dans le patron défini dans la TGS (les bits à 0 sont transparents). Le bit le plus significatif (B7) est un bit très spécial dont l'utilisation sera étudiée lors de l'étude du déplacement d'un sprite. Les bits B6, B5 et B4 sont inutilisés et doivent valoir 0.

Les 4 premiers octets de la TAS sont relatifs au sprite du plan 0, les 4 suivants au sprite du plan 1 et ainsi de suite jusqu'au plan 31. Il y a donc 128 octets dans la TAS.

REMARQUE : les deux premiers octets définissent les coordonnées du sprite par rapport au coin supérieur gauche de l'écran.

La valeur de la coordonnée verticale du point supérieur gauche a été fixée à -1 et la valeur de la coordonnée horizontale de ce coin a été fixée à 0.

Pour afficher un sprite en haut à gauche de l'écran, il faut donc mettre le premier octet relatif au plan dans la TAS à -1 et le second à 0.

Merci monsieur le constructeur !!!

2.7.3 La TGS.

La TGS est une table de 2048 octets organisée en 256 blocs de 8 octets. Elle définit le patron du sprite.

Le troisième octet de chaque entrée dans la TAS spécifie le bloc correspondant dans la TGS.

La TGS est organisée comme suit : les blocs de 8 octets sont groupés 4 par 4. Si le sprite est au format 8 X 8 ou 16 X 16 avec le bit d'agrandissement = 1, un seul bloc de 8 octets est occupé sur les 4, les 3 autres n'étant pas considérés par le VDP. Si le sprite est au format 16 X 16 avec le bit d'agrandissement à 0 ou au format 32 X 32, les 4 blocs sont visualisés dans l'ordre suivant :

Le premier bloc définit le carré supérieur gauche (8X8 pts)
Le deuxième bloc définit le carré inférieur gauche.
Le troisième bloc définit le carré supérieur droit.
Le quatrième bloc définit le carré inférieur droit.

2.7.4 Déplacement des sprites.

Nous avons vu que pour déplacer un sprite, il suffit de modifier les deux octets relatifs aux coordonnées de ce sprite dans la TAS.

Si une coordonnée a une valeur telle qu'une partie de ce sprite n'est pas dans la partie affichable de l'écran, la partie se trouvant sur l'écran est affichée normalement, celle qui est en dehors est cachée par le bord.

Ce système permet une apparition progressive des objets en mouvement. Pour faire sortir un sprite à droite ou en bas de l'écran, celui-ci doit avoir une coordonnée supérieure à 255 ou à 191 moins la taille du sprite (8 ou 16).

Pour faire apparaitre ou sortir un sprite en haut à gauche de l'écran, le problème est plus ardu.

Pour la coordonnée verticale, pas de problème, les coordonnées affichables étant comprises entre 0 et 191, il suffit de considérer l'octet de coordonnée verticale comme un octet exprimé en binaire signé si la valeur est supérieure à 191.



RAPPEL : pour déterminer une valeur en binaire signé, il suffit d'ajouter 256 à la valeur désirée ainsi, -1 s'écrit -1+256 soit 255.

Exemple : une valeur de -3 (253) pour la coordonnée verticale permettra d'afficher un sprite en partie caché par le bord supérieur de l'écran.

La coordonnée verticale peut prendre toutes les valeurs comprises entre -31 (225) et +191. En effet, il est inutile de dépasser -31, car à ce moment, un sprite de 32 X 32 points a complètement disparu de l'écran.

Pour la coordonnée horizontale, le problème est plus complexe. Les coordonnées affichables étant comprises entre 0 et 255, l'octet suffit tout juste pour définir la dite coordonnée.

L'astuce utilisée pour la coordonnée verticale n'est pas applicable. Pour réaliser ce tour de force, les concepteurs du VDP ont imaginé d'utiliser le bit le plus significatif du quatrième octet relatif au sprite concerné dans la TAS. (l'octet qui définit la couleur).

Si ce bit est à 0, il n'influence pas l'affichage.

Si ce bit est à 1, la coordonnée horizontale (contenu du deuxième octet relatif au sprite concerné dans la TAS) est diminuée de 32. La coordonnée horizontale du sprite peut donc être comprise entre -32 et +255.

REMARQUE : voyez au chapitre 7 les instructions BASE et VDP qui vous permettent de manipuler tous les registres et toutes les tables étudiés dans ce chapitre.

# 3. LE GENERATEUR SONORE AY3-8910.

## 3.1 Généralités.

Le générateur sonore AY3-8910 de GENERAL INSTRUMENTS (en abrégé PSG pour Programmable Sound Générator) est un composant relativement facile à interfacer avec n'importe quel système à microprocesseur. Il peut produire les sons les plus divers et est utilisé dans de nombreuses applications comme les synthétiseurs musicaux, les alarmes et signalisations sonores ou les MODEMS utilisant la technique FSK.

La partie sonore se fait par une conversion digitale analogique sur 4 bits, ce qui permet une grande variété d'effets.

Une des caractéristiques principales du circuit est qu'une fois ses commandes de génération sonore reçues, il produit son effet en laissant le microprocesseur libre pour continuer d'autres tâches. Ainsi, la production d'effets sonores relativement longs n'affectera en rien la vitesse d'exécution du programme qui continuera à se dérouler sans s'occuper du PSG.

Le PSG possède 3 voies mixables et permet donc la sortie de 3 sons simultanés, donc la création d'un accord musical simple majeur ou mineur.

Le PSG est un coprocesseur dont la gestion se fait au moyen de registres, ces registres sont au nombre de 16 et chacun d'entre eux va être décrit en détail au cours de ce chapitre.



## 3.2 Structure interne du PSG.

Le PSG est composé des éléments suivants :

a) Générateurs sonores : au nombre de 3, ils produisent un signal carré dont la fréquence est programmable. On les appelle canaux A, B et C. Ils n'ont pas de priorité propre et sont indépendants.

b) Générateur de bruit blanc : il produit un bruit à large spectre.

c) Mélangeur : il permet de mélanger (combiner) les sorties des 3 générateurs sonores et du générateur de bruit.

d) Contrôleur d'amplitude : il permet de sélectionner l'amplitude de sortie du signal de deux façons différentes. La première est de contrôler l'ampli--tude par le microprocesseur lui-même, elle est dite amplitude fixe. La seconde est de contrôler l'amplitude par le générateur d'enveloppes, elle est dite amplitude variable.

e) Générateur d'enveloppe : il produit une enveloppe de modulation de l'amplitude. Il possède 8 formes d'enveloppes.

f) Convertisseurs digitaux-analogiques : les 3 conver--tisseurs D/A produisent les signaux à 16 niveaux tels que le contrôleur d'amplitude les détermine.

g) Ports d'entrée/sortie : ils ne servent pas à la production sonore, ils seront analysés à la fin de ce chapitre.





### 3.3 Les différents registres du PSG.

Les registres sont au nombre de 16, numérotés R0 à R15. Les registres R14 et R15 servent à la gestion des ports d'entrée/sortie et seront analysés par la suite.

Pour produire un son, une combinaison des registres R0 à R14 doit être chargée avec des données. Chaque paramètre doit être analysé de façon à dissocier la composante bruit, la composante son, la fréquence, la forme et la durée de l'enveloppe. Une fois cette analyse effectuée, les registres peuvent être chargés et le son produit.

#### 3.3.1 Les registres R0 à R5.

Les 3 premières paires de registres (R0-R1, R2-R3, R4-R5) sont les registres de contrôle de la fréquence des 3 canaux A, B et C.
Les registres R0, R2 et R4 sont les registres de réglage fin et les 8 bits sont utilisés. Les registres R1, R3 et R5 sont les registres de réglages grossiers (seuls les 4 bits de gauche LSB sont utilisés).
Ainsi les valeurs chargées dans R0, R2 et R4 sont comprises entre 0 et 255 ; les valeurs chargées dans R1, R3 et R5 sont comprises entre 0 et 15.

La détermination de la fréquence se fait de la façon suivante: soit F la fréquence à programmer, on applique la formule suivante :

VL = 3579545 / (16 * F)

On arrondit VL à l'unité, puis on l'exprime sur 12 bits au moyen de la fonction BIN$(VL), ensuite les huit bits de droite sont transmis dans R0, R2 ou R4 et les 4 bits de gauche sont transmis dans R1, R3 ou R5. Une autre façon de procéder consiste à calculer RL = MOD(VL,256) et RH = VL\256 (c'est bien le signe \ et non pas /), il suffit alors de transmettre RL dans R0, R2 ou R4 et RH dans R1, R3 ou R5



```
Exemple si F = 440 Hz : VL = 3579545 / (16 * 440)
                        VL = 3579545 / 7040
                        VL = 508.45
On arrondit             VL = 508
On calcule              RL = MOD(508,256)
                        RL = 252
                        RH = 508\256
                        RH = 1
```

Si c'est le canal A qui doit être programmé : R0=252 et R1=1.

Détermination de F minimum et de F maximum.

Comme VL peut être exprimé sur 12 bits, la valeur de VL est comprise entre 1 et 4095. 1 donne F max. et 4095 donne F min.

```
   1=3579545/(16*Fmax) → Fmax = 3579545/16*1     = 22372
4095=3579545/(16*Fmin) → Fmin = 3579545/16*4095  =     5
```

Il est évident qu'une fréquence de l'ordre de Fmax est imperceptible par l'oreille humaine. La bande passante d téléviseur ou d'un petit ampli dépassant rarement 5000 Hz nous retiendrons cette fréquence comme valeur maximum à produire.

Un simple calcul donne VL= 44.

Donc les valeurs de VL seront comprises entre 44 et 4095.

#### 3.3.2 Le registre R6.

Le registre R6 détermine la fréquence du générateur de br seuls les 5 bits les moins significatifs sont utilisés. valeur de R6 est donc comprise entre 1 et 31. La même formule que pour R0-R5 est utilisée, un simple calcul no donne donc la fréquence du générateur de bruit entre 2237 et 7216 Hz.

#### 3.3.3 Le registre R7.

Le registre R7 contrôle le mélange entre les 3 générateur sonores et le générateur de bruit. R7 sert aussi au cont des 2 ports dont nous parlerons par la suite.

Voici un tableau résumant les effets du registre R7.

| BIT | = 0 | = 1 |
|---|---|---|
| 7 | PORT B ENTREE | PORT B SORTIE |
| 6 | PORT A ENTREE | PORT A SORTIE |
| 5 | BRUIT SUR CANAL C ON | BRUIT SUR CANAL C OFF |
| 4 | BRUIT SUR CANAL B ON | BRUIT SUR CANAL B OFF |
| 3 | BRUIT SUR CANAL A ON | BRUIT SUR CANAL A OFF |
| 2 | SON SUR CANAL C ON | SON SUR CANAL C OFF |
| 1 | SON SUR CANAL B ON | SON SUR CANAL B OFF |
| 0 | SON SUR CANAL A ON | SON SUR CANAL A OFF |

NOTE : Mettre un canal sur OFF ne suffit pas pour arrêter l'émission de celui-ci ; il faut écrire un 0 dans le registre de contrôle d'amplitude (voir ci-dessous).

Exemple : Je veux sur le canal A du son et pas de bruit, sur le canal B du bruit et du son et sur le canal C du bruit uniquement.

```
Valeur : x x 0 0 1 1 0 0 =  12
   bit : 7 6 5 4 3 2 1 0
         (x x) = SANS IMPORTANCE
         Il suffit d'écrire 12 dans le registre 7.
```

### 3.3.4 Les registres R8 à 10.

Les registres R8 à R10 contrôlent les amplitudes des canaux A, B et C, seuls les 4 bits les moins significatifs sont utilisés donc les valeurs possibles sont comprises entre 0 et 15. 0 signifie que l'amplitude est minimum (nulle) et 15 correspond à l'amplitude maximum. Le cinquième bit (BIT4) est le bit de sélection du mode de fonctionnement du contrôle de l'amplitude. Si BIT4 est 0 l'amplitude ne varie pas, si BIT4 est 1 l'amplitude est contrôlée par le générateur d'enveloppe (voir ci-dessous).

### 3.3.5 Les registres R11 et R12.

Ces deux registres contrôlent la période de l'enveloppe. Un calcul avec une formule similaire à celle utilisée pour R0-R5 est effectué pour déterminer la valeur de R11 et R12.



Formule : VL=3579545*P/256 où P est la période de l'enve-
loppe.

Les 8 bits des registres R11 et R12 sont utilisés, donc la valeur de VL est comprise entre 0 et 65535.

Un calcul similaire à celui effectué pour les registres R
à R5 nous permet de déterminer Pmin et Pmax.

Pmin = 1*256/3579545 = 0,0000715 seconde.
Pmax = 65535*256/3579545 = 4,6868973 secondes.

### 3.3.6 Le registre R13.

Le registre R13 contrôle la forme de la modulation utili
Si le BIT4 décrit dans les registres R8 à R10 est 1, la modulation a lieu sinon la programmation du registre 13
ignorée.

Seuls les 4 bits les moins significatifs sont utilisés.

Table des modulations :

| BIT 3 2 1 0 | | FORME DE L'ENVELOPPE | VALE POSSI |
|---|---|---|---|
| 0 0 X X | A | Un seul cycle, commence avec une amplitude maximum qui diminue pour devenir nulle. | 0,1, |
| 0 1 X X | B | Un seul cycle commence avec une amplitude nulle qui augmente pour atteindre sa valeur maximum, ensuite retombe brusquement à 0. | 4,5, |
| 1 0 0 0 | C | Comme A mais se répète sans cesse. | 8 |
| 1 0 1 0 | D | Comme C mais remonte de façon plus marquée vers le maximum (ATTACK). | 10 |
| 1 0 1 1 | E | Comme A mais revient ensuite au maximum et y reste. | 11 |
| 1 1 0 0 | F | Comme B mais se répète sans cesse. | 12 |
| 1 1 0 1 | G | Comme B mais reste au maximum. | 13 |
| 1 1 1 0 | H | Comme F mais avec une attaque plus marquée. | 14 |

## 3.4 Utilisation des registres R0 à R13 (programmation).

La programmation peut se faire de deux façons différentes en BASIC, soit en utilisant l'instruction SOUND, soit en utilisant l'instruction OUT. Cette deuxième manière de procéder est aussi valable en assembleur.

La programmation d'un son au moyen de la commande SOUND est très aisée, il suffit d'écrire SOUND NR,VL où NR est le numéro du registre (compris entre 0 et 13) et VL est la valeur à écrire dans ce registre (comprise entre 0 et 255).

La programmation au moyen de l'instruction OUT est un peu plus compliquée, il faut donner le numéro du registre sur le port A0H (160) et ensuite écrire la valeur de VL sur le port A1H (161).

L'instruction SOUND NR,VL peut s'écrire OUT 160,NR : OUT 161,VL.

Programmes de démonstration:

A) Génération d'une explosion (coup de feu).

```
10 SOUND 6,15
20 SOUND 7,7
30 SOUND 8,16
40 SOUND 9,16
50 SOUND 10,16
60 SOUND 12,16
70 SOUND 13,0
80 FOR I=1 TO 500 : NEXT I
90 GOTO 70
```

explication :

La ligne 10 détermine la fréquence du bruit (R6).
La ligne 20 valide la sortie du bruit sur les canaux A,B et C.
Les lignes 30 à 50 sélectionnent la modulation des 3 canaux au moyen du générateur d'enveloppe.
La ligne 60 détermine la période de l'enveloppe.
La ligne 70 determine la forme de l'enveloppe (forme A) et produit le son. Chaque instruction SOUND 13,0 reproduira le même son ; c'est l'objet des lignes 80 (délai) et 90.



Au moyen de l'instruction OUT, le programme s'écrit :

```
10 LA=160 : WR=161 ; REM LA pour Latch Adress et
                         WR pour Write Reg.
20 OUT LA,6 : OUT WR,15
30 OUT LA,7 : OUT WR,7
40 OUT LA,8 : OUT WR,16
50 OUT LA,9 : OUT WR,16
60 OUT LA,10: OUT WR,16
70 OUT LA,12: OUT WR,16
80 OUT LA,13: OUT WR,0
90 FOR I=1 TO 500 : NEXT I
95 GOTO 80
```

B) Génération d'un bruit de sirène.

```
10 SOUND 7,62 ; ' SON CANAL A ON
20 SOUND 8,15 ; ' VOLUME MAX SUR CANAL A
30 FOR J=1 TO 3
40 FOR I=100 TO 200 ; ' VALEUR DE LA FREQUENCE
50 SOUND 0,I ; ' REGISTRE FREQUENCE CANAL A
60 NEXT I
70 FOR I=200 TO 100 STEP -1 ; ' ON REDESCEND
80 SOUND 0,I
90 NEXT I
95 NEXT J ; ' ON FAIT 3 FOIS PIN-PON
99 OUT 8,0 ; ' AMPLITUDE A 0, DONC ARRET DU SON.
```

Pour les amateurs de programmation en ASSEMBLEUR nous avons relevé une sous routine dans la ROM interne qui facilite la programmation du PSG. Elle se trouve à l'adresse hexadécimale 1102. En voici un désassemb--lage obtenu avec le programme BASIC de l'annexe.

```
1102  F3      DI
1103  D3 A0   OUT   (0A0H),A
1105  F5      PUSH  AF
1106  7B      LD    A,E
1107  D3 A1   OUT   (0A1H),A
1109  FB      EI
110A  F1      POP   AF
110B  C9      RET
```

Pour l'utiliser, il suffit de mettre le numéro du registr dans l'accumulateur A et la valeur dans le registre E. Ensuite, faire un CALL à 1102H. Cette routine ne modifie aucun registre.

```
EXEMPLE : LD   E,62    ; E = 62
          LD   A,7     ; sélectionne le registre 7
          CALL 1102H   ; appel à la ROM
```

## 3.5 Utilisation des ports d'entrée/sortie.

### 3.5.1 Préliminaires.

Maintenant que l'essentiel sur le générateur sonore a été dit, attaquons nous aux deux ports d'entrée/sortie. Nous avons vu lors de l'étude du registre R7 que les bits 7 et 6 règlent le sens de la transmission des données (0=entrée, 1=sortie). Les registres R14 et R15 sont utilisés pour l'écriture et la lecture de ces 2 ports.

### 3.5.2 Ecriture et lecture des ports d'entrée/sortie.

L'écriture dans les ports d'entrée/sortie se fait exactement comme l'écriture dans un registre du généra--teur sonore, mais l'instruction BASIC SOUND n'est plus utilisable, il faut donc utiliser l'instruction OUT.

Les deux ports sont appelés port A et port B, l'écriture dans le port A se fait en chargeant R14 avec la valeur à écrire par la suite d'instructions suivantes :

```
OUT 160,14 : OUT 161,VL
```

L'écriture dans le port B se fait en chargeant R15 avec la valeur à écrire.

```
OUT 160,15 : OUT 161,VL
```

Ce qui a été dit au sujet de l'assembleur lors de l'étude de la programmation de R0 à R13 reste valable pour la programmation de R14 et R15.

Passons maintenant à la lecture.
Pour lire le contenu d'un port (A ou B), il faut charger le port 160 du microprocesseur avec 14 pour le port A ou 15 pour le port B, ensuite effectuer une lecture du port 162 du microprocesseur au moyen de la fonction BASIC INP.

EXEMPLE : lecture du contenu du port A.

```
OUT 160,14 :X=INP(162)
```

A la suite de ces instructions, la variable X contient le contenu du port A.



### 3.5.3 Contenu des ports A et B du PSG.

Voyons enfin l'utilisation de ces deux ports.

Le port A sert principalement à la lecture des JOYSTICKS ou manettes de jeu.

L'annexe D du présent volume donne la table de découpage des bits du port A en fonction du numéro de la manette de jeu et de la position de celle-ci.

Les 4 bits de poids faible sont utilisés pour déterminer le sens de déplacement de la manette de jeu.

Le bit 0 passe à 0 si la manette est poussée vers le haut
Le bit 1 " " " " " " " " " le bas
Le bit 2 " " " " " " " " " la gauc
Le bit 3 " " " " " " " " " la droi
Le bit 4 detecte la pression sur le bouton de tir
Le bit 5 est utilisé pour armer le TRIGGER pour les manettes analogiques.
Le bit 6 n'est pas utilisé en version Européenne.
Le bit 7 sert à la lecture de cassette.

Le port 15 est utilisé principalement pour sélectionner la manette 1 ou la manette 2 et pour la gestion des palettes analogiques (bit 0 à 6). Le bit 7 n'est pas utilisé en version Européenne.

### 3.5.4 Routines assembleur.

Il existe des routines assembleur qui permettent la lecture des manettes logiques, des boutons de tir et des manettes analogiques.

A) lecture des manettes logiques.

Il suffit de mettre dans A le nombre (1 ou 2) correspondant au numéro de la manette ou 0 pour le clavier et de faire un CALL à l'adresse 00D5H. La valeur du joystick est rendue dans A.

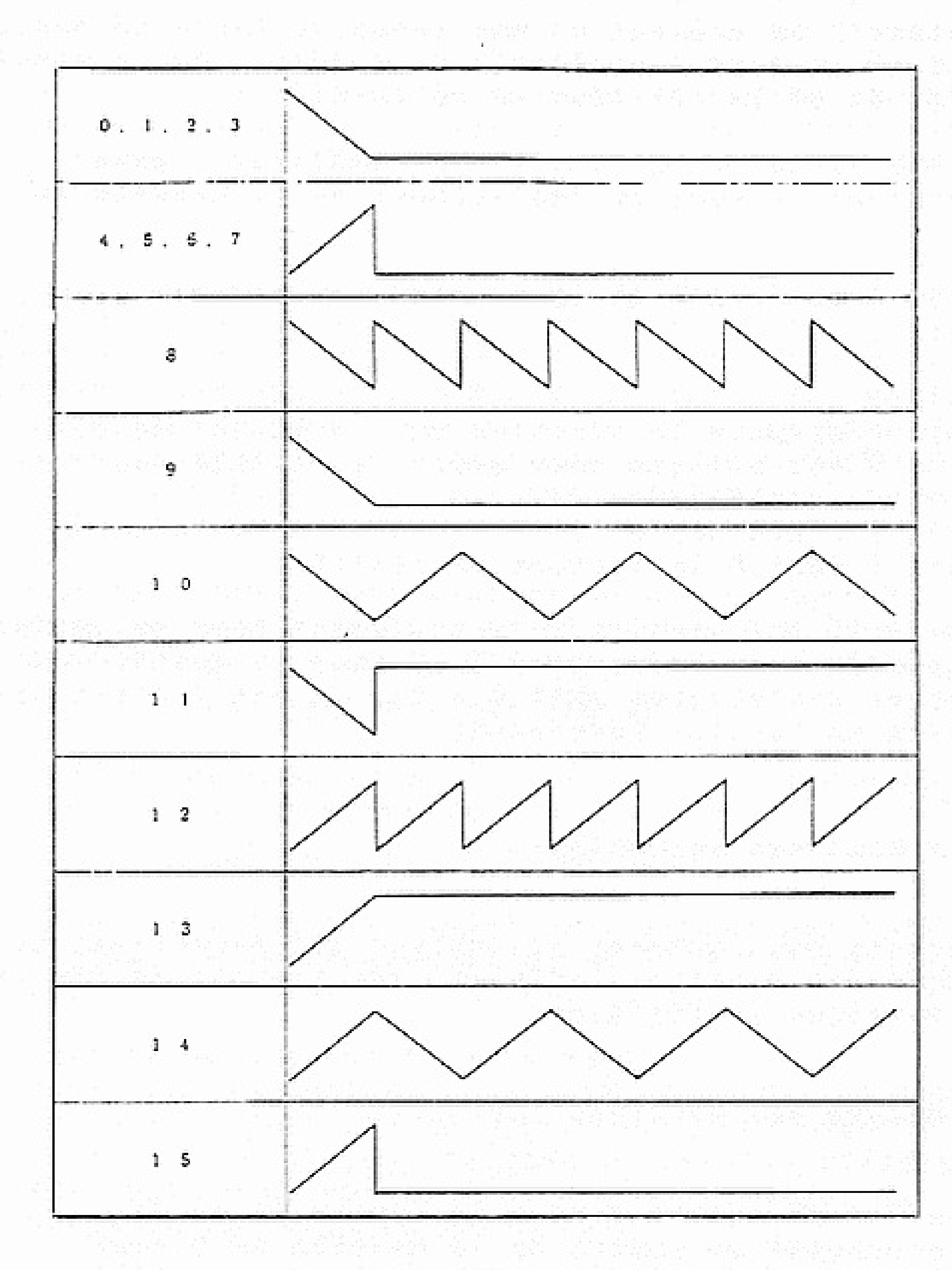
B) lecture du bouton de tir.

Il suffit de mettre dans A le numéro de la manette et de faire un CALL à l'adresse 00D8H.

C) Lecture des manettes analogiques.

Deux routines existent : une en 00DBH qui réalise l'équivalent de la fonction PAD et une en 00DEH qui réalise l'équivalent de la fonction PDL.

------------------




Les différentes formes du signal sonore du PSG



# 4. LE PPI (Programmable Port Interface).

## 4.1 Généralités.

Le PPI est un circuit fabriqué par INTEL sous la dénomination 8255A, c'est un circuit d'interfaçage prévu pour les processeurs de la famille du 8080.

Il possède 24 bits d'entrée/sortie qui peuvent être programmés en 2 groupes de 12 bits et utilisés dans 3 modes principaux.

Dans le premier mode (mode 0), chaque groupe de 12 bits peut être programmé par tranche de 4 bits en entrée comme en sortie.

Dans le second mode (mode 1), chaque groupe de 12 bits peut être programmé de la façon suivante : 8 bits sont utilisés en entrée/sortie, les 4 autres sont utilisés pour le HANDSHAKING (contrôle de la transmission).

Le troisième mode (mode 2), est un mode où 8 bits sont utilisés comme PORT bidirectionnel et 5 bits pour le HANDSHAKING.

Le PPI possède aussi la possibilité de positionner des bits à l'état 1 ou 0 directement.

Pour plus de facilité, le PPI est divisé en 3 ports de 8 bits distincts appelés PORT A, PORT B et PORT C.

Le PORT C se divise en 2 groupes de 4 bits pour former les groupes de 12 bits avec A et B.

## 4.2 Découpage et utilisation des PORTS A, B et C.

### 4.2.1 Le PORT A.

Le PORT A est utilisé pour la commutation des SLOTS mémoires.

La mémoire est divisée en 4 banks de 16 K.

| | | |
|---|---|---|
| BANK 0 | adresse | 0000 à 3FFF |
| BANK 1 | " | 4000 à 7FFF |
| BANK 2 | " | 8000 à BFFF |
| BANK 3 | " | C000 à FFFF |

Sur chaque BANK, on peut sélectionner un SLOT parmi 4.

Les 8 bits du port A sont divisés de la façon suivante :

| | |
|---|---|
| BITS 0 et 1 | sélection du SLOT pour le BANK 0 |
| BITS 2 et 3 | " " " " " " 1 |
| BITS 4 et 5 | " " " " " " 2 |
| BITS 6 et 7 | " " " " " " 3 |

Chaque série de 2 bits permet 4 combinaisons : 00, 01, 10 et 11.

En BASIC, dans une configuration standard, tous les bits sont à 0 (les slots 0 sont sélectionnés).

### 4.2.2 Le PORT B.

Le port B est très important car c'est lui qui permet la lecture du clavier. Les 8 bits sont utilisés unique--ment en lecture pour déterminer si une touche est pressée ou non.

Nous nous attarderons par la suite sur le port B lors de l'analyse du fonctionnement du clavier.



### 4.2.3 Le PORT C.

Le port C est divisé en 2 blocs de 4 bits numérotés B0-B3 et B4-B7.

Les bits B0-B3 sont utilisés uniquement en lecture et permettent de fournir l'adresse de la ligne des touches du clavier à lire.

Nous nous attarderons aussi par la suite sur le bloc B0-B3 du port C lors de l'analyse du fonctionnement du clavier.

Les bits B4-B7 sont utilisés pour la cassette et pour le signal SOUND.

Le bit B4 commande le démarrage ou l'arrêt de la cassette (signal CASON).

Le bit B5 commande l'écriture sur la cassette (signal CASWR ).

Le bit B6 commande l'allumage de la lampe CAPS.

Le bit B7 commande le signal SOUND.

## 4.3 Programmation du PPI.

### 4.3.1 Introduction.

Le PPI est interfacé aux adresses suivantes :

a) A8H 168 : lecture et écriture du PORT A.
b) A9H 169 : lecture du PORT B.
c) AAH 170 : lecture et écriture du PORT C.
d) ABH 171 : écriture du registre de contrôle.

REMARQUE : de la configuration matérielle nous pouvons déduire que le PORT B est utilisé en lecture uniquement, le registre de contrôle est utilisé en écriture uniquement et les PORTS A et C sont utilisés dans les deux modes.

Des 3 modes décrits brièvement dans les généralités, seul le MODE 0 sera étudié car il suffit à toutes les manipulations envisagées.

Le PPI est programmable à travers un registre de contrôle dans lequel on ne peut qu'écrire. Aucune lecture de ce registre n'est permise.

Les PORTS du PPI doivent être divisés en deux groupes :

Le groupe A composé du PORT A et les 4 bits de poids fort du PORT C (B4-B7).

Le groupe B composé du PORT B et les 4 bits de poids faible du PORT C (B0-B3).

Le groupe B est entièrement réservé au clavier.

Le groupe A s'occupe de la gestion des SLOTS, de la cassette, du son et de la touche CAPS.

### 4.3.2 Programmation.

A) Ecriture dans le registre de contrôle.

On écrit dans le registre de contrôle par une simple instruction OUT sur le PORT ABH du processeur.



Le mot de contrôle est un mot de 8 bits dont voici la signification bit par bit.

BIT 7 : toujours 1 si c'est un mot de contrôle.

BIT 6 : détermination du mode de fonctionnement du groupe A, pour sélectionner le MODE 0, ce bit doit être 0. S'il est à l'état 1, il sélectionne le MODE 2.

BIT 5 : détermination du mode de fonctionnement du groupe A, pour sélectionner le MODE 0, ce bit doit être 1. S'il est à l'état 1, il sélectionne le MODE 1.

BIT 4 : détermination du sens de fonctionnement du PORT A 0 signifie en SORTIE et 1 signifie en ENTREE. Sera toujours 1.

BIT 3 : détermination du sens de fonctionnement de la partie haute du PORT C, 0 signifie en SORTIE et 1 signifie en ENTREE.

BIT 2 : détermination du mode de fonctionnement du groupe B, 0 signifie MODE 0 et 1 signifie MODE 1. Sera toujours 0.

BIT 1 : détermination du sens de fonctionnement du PORT B 0 signifie en SORTIE et 1 en ENTREE. Sera toujour 1.

BIT 0 : détermination du sens de fonctionnement de la partie basse du PORT C. 0 signifie en SORTIE et 1 signifie en ENTREE. Sera toujours 0.

Si le BIT 7 est égal à 0, le registre n'est plus utilisé en tant que contrôleur des PORTS mais il permet de posi-tionner les bits du PORT C à 1 ou à 0.

BIT 7 = 0 : fonctionnement en positionnement de bit.

BIT 6, BIT 5 et BIT 4 : non utilisés.

BIT 3, BIT 2 et BIT 1 : donnent le numéro du bit à posi-tionner.
EXEMPLE : pour positionner le BI mettre 1 dans B3, 0 dans B2 et 1 dans B1 car 101 donne 5.

BIT 0 : donne le sens du positionnement, 1 signifie positionnement du bit à 1 et 0 signifie position-nement du bit à 0.

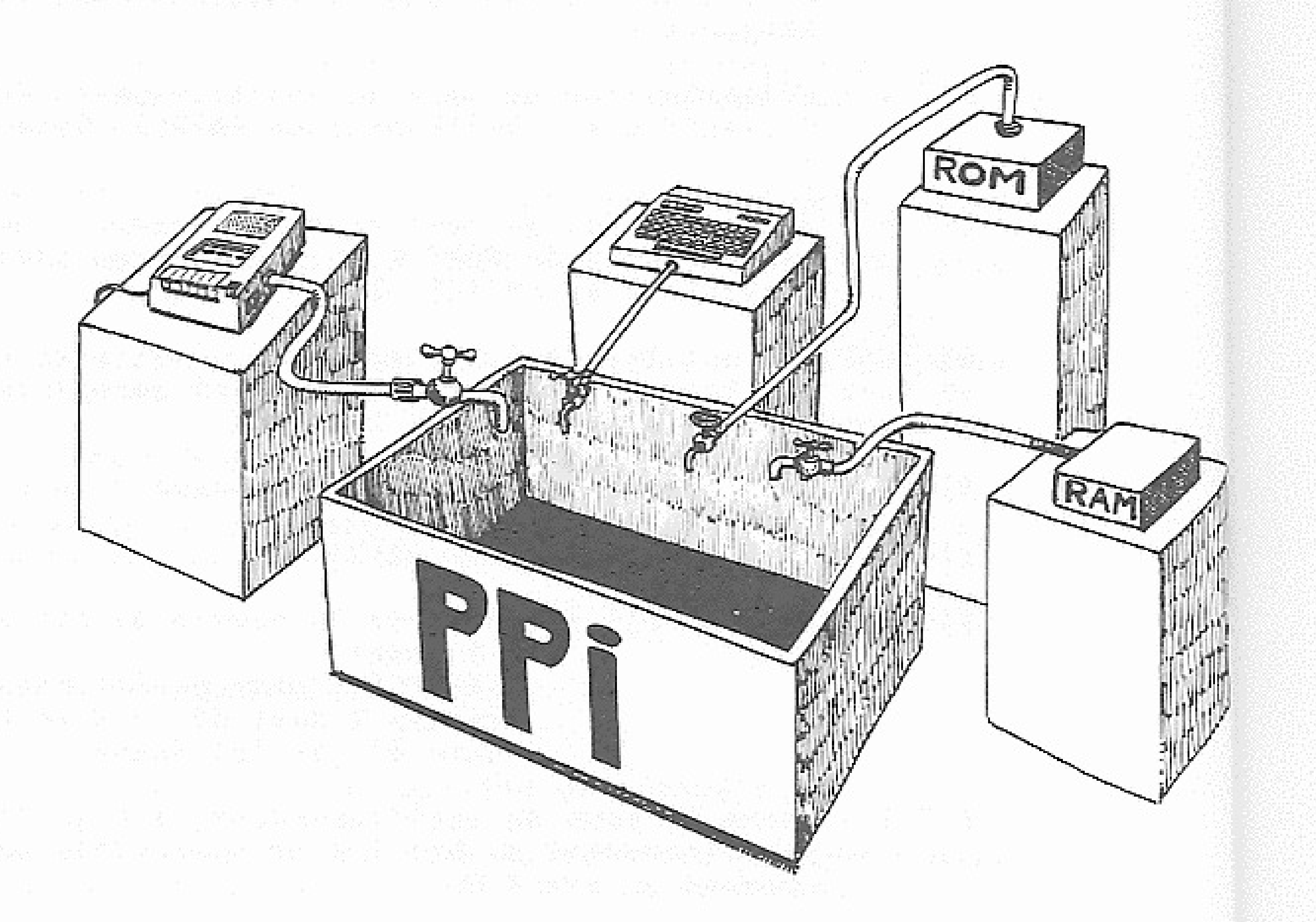
La programmation se fait donc en envoyant le mot d'état convenable sur le registre de contrôle et en effectuant une lecture ou une écriture sur le PORT idoine.

EXEMPLE : pour démarrer le moteur de la cassette il suffit de mettre le BIT 4 du PORT C à 0.

```
10 OUT &HAB,&B00001000
```

Nous avons utilisé le mode de positionnement des bits, analysons l'octet 00001000.

Le premier bit en partant de la gauche (B7), égal à 0, signifie que l'on est en mode de positionnement de bits. Les 3 bits suivants (B6 à B4), ne servent à rien. Les bits B3 à B1 valent 100, c'est-à-dire 4. Donc c'est le bit B4 du PORT C qui est sélectionné. Le bit B0 vaut 0, ce qui signifie mettre le bit B4 à 0.






## 4.4 Gestion du clavier.

Le clavier est comparable à une matrice de 9 lignes et de 8 colonnes, elle permet donc de disposer 72 touches selon le schéma suivant :

| PORT B = | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
|---|---|---|---|---|---|---|---|---|
| PORT C | | | | | | | | |
| LIGNE 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| LIGNE 1 | 8 | 9 | - | + | \ | [ | ] | ; |
| LIGNE 2 | " | £ | < | > | ? | ACC | A | B |
| LIGNE 3 | C | D | E | F | G | H | I | J |
| LIGNE 4 | K | L | M | N | O | P | Q | R |
| LIGNE 5 | S | T | U | V | W | X | Y | Z |
| LIGNE 6 | SFT | CTL | GRAPH | CAP | CODE | F1 | F2 | F3 |
| LIGNE 7 | F4 | F5 | ESC | TAB | STOP | BS | SEL | RC |
| LIGNE 8 | SPC | CLS | INS | DEL | ← | ↑ | ↓ | → |

REMARQUE : une seule des gravures des touches a été représentée.

Les abréviations suivantes ont été utilisées :

ACC = touche accent à côté de l'accolade droite.
SFT = les deux touches SHIFT.
CTL = touche CTRL
SEL = touche SELECT.
RC = touche ENTER.
SPC = barre d'espacement.

Pour lire cette matrice, on transmet sur la partie basse du PORT C (B0-B3) le numéro de la ligne à scruter. Ensuite, on lit le contenu du port B.

Un bit est à 0 si la touche correspondante est enfoncée, sinon il est à 1.

Les bits du PORT B étant numérotés de B0 à B7, si l'on appuye sur la touche A par exemple, le bit B6 passe à 0; les autres bits restent à 1.

Le bit B6 correspondant à la touche A est lisible unique--ment lorsque le numéro de la ligne (en l'occurence 2) a été inscrit sur le PORT C.

EXEMPLE : pour décoder l'appui sur la touche SELECT (non utilisé par BASIC), il faut :

1-écrire le numéro de la ligne dans le PORT C (ligne 7).
2-lire le contenu du PORT B.
3-masquer les bits inutiles.
4-tester le bit concerné (B6).

C'est le but du petit programme suivant :

```
10 OUT &HAA,7    : ' écriture du numéro de ligne dans C.
20 X=INP(&HA9)   : ' lecture du PORT B.
30 X=X AND 64    : ' masquage des bits différents de B6.
40 IF X=0 THEN PRINT "TOUCHE SELECT ENFONCEE"
50 GOTO 10       : ' on recommence le test.
```

Vous pouvez, par la même méthode, décoder l'appui sur une touche quelconque sans passer par l'INPUT ou l'INKEY$.

En assembleur, le décodage se fait de la même façon. Bien sûr, nous verrons qu'une telle routine assembleur existe déjà dans la ROM BASIC, comment l'utiliser et même l'intercepter pour transformer un clavier QWERTY en AZERTY. Mais ça, c'est l'objet d'un autre chapitre.

Enfin, pour les impatients, voici une petite routine assembleur de test d'une touche (en l'occurence, la touche SELECT).

```
LD      A,8
OUT     (AAH),A
IN      (A9H)
AND     40H
```

Suivent alors le test, si A est 0, et le traitement.



# 5 STRUCTURE INTERNE DE LA ROM MSX

## 5.1 Généralités.

La ROM MSX contient un système d'exploitation très rudimentaire, un interpréteur BASIC et un ensemble de routines de gestion des périphériques.

Le propos de ce chapitre est de décrire les opérations fondamentales de la ROM pour que vous puissiez en tirer le meilleur parti lors de la programmation en ASSEMBLEUR.

Un ordinateur sans système d'exploitation présente peu d'intérêt.
Le système d'exploitation permet la communication entre l'utilisateur et la machine, ce qui signifie : lire le clavier pour "voir" si on appuye sur une touche et écrire des messages sur l'écran.

Lorsque nous écrivons un programme, il y a un programme dans l'ordinateur qui reçoit nos ordres d'écriture. C'est le système d'exploitation.

5.2 Composition de la ROM MSX

La ROM se compose de :

1) Un certain nombre de DRIVERS (programmes d'interfaçage) pour chacun des périphériques comme le clavier, l'écran, la cassette et l'imprimante (BIOS).

2) L'interpréteur syntaxique du langage BASIC.

3) Des routines mathématiques et arithmétiques.

4) Des routines de gestion de la mémoire et des tables.

5) Un moniteur, programme qui consulte continuellement le clavier en attente d'une entrée.

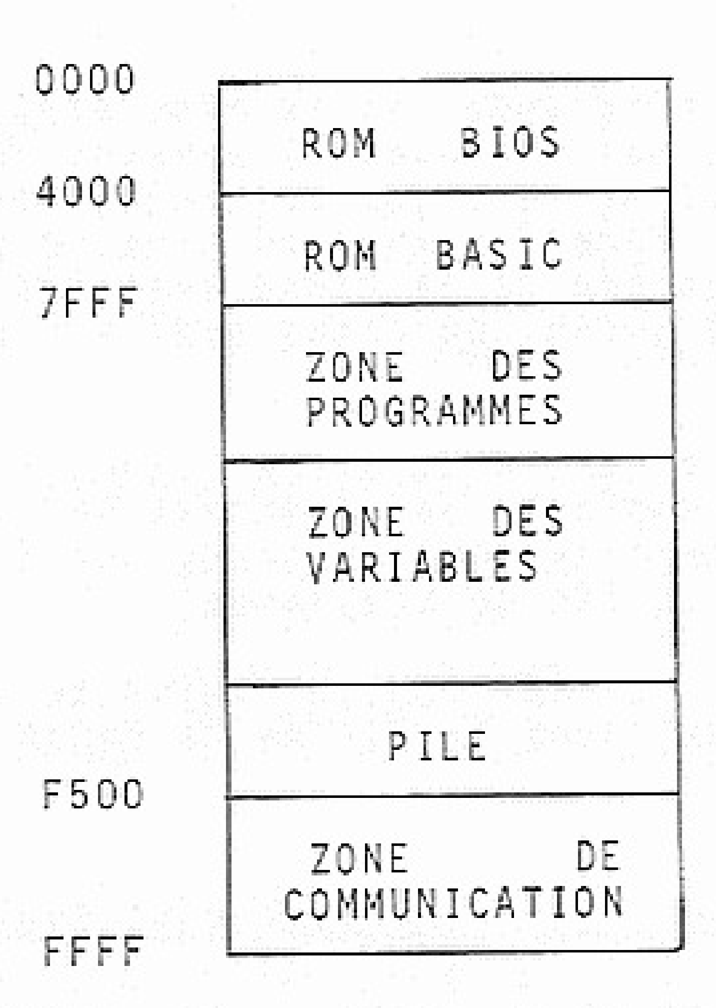
6) Divers utilitaires comme l'éditeur pleine page, le producteur de LIST, etc...

7) Un ensemble de tables (générateur de caractères, conversion de clavier,...).



5.3 Utilisation de la mémoire.

La mémoire est découpée de la façon suivante :




La zone pour les programmes et les variables peut être divisée en deux tables principales :

1° La TIP : Table des Instructions du Programme.

2° La TV : Table des Variables.

Insérer ou effacer une ligne BASIC d'un programme produit un accroissement ou une réduction de la TIP, de la même façon, définir une nouvelle variable accroît la taille de la TV. Comme les adresses de début des tables sont variables, elles sont définies à un endroit fixe de la région de communication. Ce principe permet de déplacer les tables où l'on veut et de toujours savoir où elles sont situées.

La TV contient le nom et la valeur de chaque variable contenue dans le programme. Elle est divisée en 2 sous-tables définies en fonction du type de variable.

## 5.4 Structure de la Table des Instructions de Programme (TIP).

La TIP contient les lignes de programmes BASIC, elle commence en 8001H dans les systèmes 32 K et en C001H dans les systèmes 16 K.

Toutes les lignes d'un programme ont la même structure. On trouve d'abord 2 octets qui indiquent l'adresse réelle en mémoire du premier octet de la ligne suivante, ensuite on trouve le numéro de la ligne courante en binaire sur 2 octets, ensuite on trouve le contenu de la ligne avec les mots clés sous forme de codes, et enfin, un octet égal à 0 pour indiquer la fin d'une ligne.

A la fin du programme, après la dernière ligne, l'adresse de la ligne suivante est remplacée par 2 octets qui valent 00. Un programme BASIC se termine donc toujours par 3 octets 00 (1 de fin de ligne + 2 de fin de programme).

Exemple : Le programme : 10 A$="COUCOU" suivi de 20 B$="TEST" est stocké en hexadécimal dans la TIP de la façon suivante (Version 32 K RAM)

| Adresse | cont | explication | Adresse | cont | explication |
|---|---|---|---|---|---|
| 8001 | 11 | 8011 ADRESSE DE LA | 8011 | 1F | 801F ADRESSE DE LA |
| 8002 | 80 | LIGNE SUIVANTE | 8012 | 80 | LIGNE SUIVANTE |
| 8003 | 0A | 10 NUMERO DE LA | 8013 | 14 | 20 NUMERO DE LA |
| 8004 | 00 | LIGNE COURANTE | 8014 | 00 | LIGNE COURANTE |
| 8005 | 41 | A (CORPS DE LA | 8015 | 42 | B (CORPS DE LA |
| 8006 | 24 | $ LIGNE) | 8016 | 24 | $ LIGNE) |
| 8007 | F1 | = | 8017 | F1 | = |
| 8008 | 22 | " | 8018 | 22 | " |
| 8009 | 43 | C | 8019 | 54 | T |
| 800A | 4F | O | 801A | 45 | E |
| 800B | 52 | U | 801B | 53 | S |
| 800C | 43 | C | 801C | 54 | T |
| 800D | 4F | O | 801D | 22 | " |
| 800E | 52 | U | 801E | 00 | FIN DE LIGNE |
| 800F | 22 | " | 801F | 00 | FIN DE |
| 8010 | 00 | FIN DE LIGNE | 8020 | 00 | PROGRAMME |



## 5.5 Structure de la Table des Variables (TV).

Cette table contient toutes les variables définies dans l
programme BASIC. Elle est divisée de façon interne en 2
parties, la première contient des informations sur toutes
les variables simples (non dimensionnées) et la deuxième
contient des informations sur toutes les variables dimen-
sionnées.

Comme la TIP, la TV se trouve en RAM et il y a 1 pointeur
pour chacune des 2 parties dans la région de communicati
l'adresse F6C2H contient l'adresse de la table des variab
simples, et l'adresse F6C4H contient l'adresse de la tabl
des variables dimensionnées.

La TV commence en général à la fin de la TIP.

Quelle que soit la partie de la table concernée, les 3
premiers octets de chaque entrée dans la table ont le mêm
format. Le premier indique le type de variable (2,3,4,8
voir format des variables). Le deuxième et le troisième
représentent les 2 premiers caractères du nom de la varia
ble.

Le type de variable détermine la longueur de la zone qui
la représente, ainsi une variable entière non dimensionné
possède une entrée dans la TV de longueur 5 : 2 (type) +
(les 3 premiers).

Dans la deuxième partie de la table, on retrouve les var
dimensionnées. Les entrées de cette partie de table ont
mêmes 3 premiers octets que les variables simples, mais
sont suivis de 2 octets qui représentent l'OFFSET à ajou
pour trouver la variable suivante (autrement dit, ils con
tiennent : 3 + (dim+1) * TYPE où dim est la dimension de
variable et TYPE le code du type de variable). L'octet
suivant indique le nombre d'index, et les 2 octets suivan
la dimension de la variable + 1 (pour tenir compte de l'
0).

Les variables sont disposées dans la TV au fur et à mesur
leur apparition dans le programme, il n'y a pas d'ordre a
bétique. Les variables dimensionnées sont souvent déplac
car l'apparition d'une nouvelle variable non dimensionnée
produit un déplacement total de la zone des variables dim
sionnées.

Exemple :
```
10 DIM A(5)
20 PRINT VARPTR(A(1))
30 B=VARPTR(A(1))
40 PRINT B
```

Cet exemple produira 2 valeurs différentes pour l'adresse de la variable A(1), car entre la ligne 20 et la ligne 40, une nouvelle variable simple (B) est apparue.

Les variables multidimension sont mémorisées dans l'ordre des colonnes, de cette façon, un déplacement de l'index gauche est plus rapide qu'un déplacement de l'index droit.

Exemples de stockage de différents types de variables.

Variables simples

A%=100

| | | |
|---|---|---|
| Code | 02 | variable entière |
| Nom | 41 | code ASCII de A |
| | 00 | pas de 2° lettre |
| valeur | 64 | valeur = 100 |
| | 00 | |

BE$="COUCOU"

| | | |
|---|---|---|
| Code | 03 | variable chaîne |
| NOM | 42 | code ASCII de B |
| | 45 | code ASCII de E |
| valeur | 93 | pointeur vers la |
| | D0 | qui contient COUCOU |

AZ#=123456789

| | | |
|---|---|---|
| Code | 08 | variable DP |
| Nom | 41 | A |
| | 5A | Z |
| VALEUR | 49 | Exposant |
| | 12 | valeur sur 7 octets |
| | 34 | |
| | 56 | |
| | 78 | |
| | 90 | |
| | 00 | |
| | 00 | |

Variables à une dimension

DIM A%(5)

| | | |
|---|---|---|
| Code | 02 | variable entière |
| Nom | 41 | code ASCII de A |
| | 00 | pas de 2° lettre |
| Offset | 0F | pour la variable |
| | 00 | suivante |
| N dim | 01 | nombre de dimension |
| n élé | 06 | DIM + 1 |
| Valeurs | 00 | sur 2 octets |
| | yy | valeur A(0) |
| | yy | |
| | zz | valeur A(1) |
| | zz | |
| | :: | ........... |
| | ww | valeur A(5) |
| | ww | |

Début de la variable suivante



5.6 <u>Espace réservé à la pile d'adresses.</u>

Il existe un espace réservé pour les adresses de retour de sous-routines. C'est l'espace réservé à la pile d'adresses.

Une instruction CALL ou RST sauvegarde son adresse de retour dans cet espace et avance le pointeur (SP) de deux octets. Une instruction RET recule le pointeur de deux octets.

Cet espace est surtout utilisé par les instructions FOR NEXT et GOSUB. Celles-ci poussent un certain nombre d'octets dans la pile afin de pouvoir les dépiler ensuit

Avant chaque nouvelle allocation d'espace, la routine de gestion de mémoire effectue un test afin de déterminer s'il reste un espace mémoire suffisant, c'est à dire, si la PILE et la TV ne vont pas interférer l'une dans l'autr Si la place est insuffisante, le système produit le messa ge d'erreur suivant : OUT OF MEMORY.

Toutes les variables associées à une boucle FOR sont transmises dans la pile jusqu'à la fin de la boucle. Quar une instruction NEXT est rencontrée, une recherche est effectuée dans la pile pour retrouver une FRAME (suite d'octets) qui porte sur la même variable index. Si cette FRAME n'est pas trouvée, un message 'NEXT WITHOUT FOR' es produit.

Format de la FRAME de l'instruction FOR.

| | | | |
|---|---|---|---|
| Bas de mémoire | : | Code de l'instruction FOR | : 1 octe |
| | : | Adresse de la variable index | : 2 octe |
| | : | Signe de l'incrément (+ou-) | : 1 octe |
| | : | Valeur de l'incrément | : 4 octe |
| | : | Valeur de l'arrêt (TO) | : 4 octe |
| | : | Numéro de ligne du FOR | : 2 octe |
| Haut de mémoire | : | Adresse du FOR dans le texte | : 2 octe |
| | | Total | 16 octe |

Lors de l'apparition d'une instruction GOSUB, une FRAME est poussée dans la pile, et lors de l'apparition d'une instruction RETURN, la pile est fouillée pour retrouver la FRAME du GOSUB la plus proche. S'il n'y a pas de FRAME, un message 'RETURN WITHOUT GOSUB' est produit.

Format de la FRAME de l'instruction GOSUB.

| | | | |
|---|---|---|---|
| Bas de mémoire | : | Code de l'instruction GOSUB | : 1 octet |
| | : | Numéro de la ligne du GOSUB | : 2 octets |
| Haut de mémoire | : | Adresse ligne dans la TIP | : 2 octets |
| | | Total | 5 octets |



5.7 <u>L'espace réservé aux chaînes.</u>

Cet espace est réservé aux chaînes ayant subi une transformation (concaténation, LEFT$, MID$, RIGHT$,...). Les chaînes simples sont stockées directement dans la Table des Instructions de Programme.

Exemple : dans le programme :

```
10 A$="COUCOU"
20 B$="IL FAIT FROID"
30 C$=A$+B$
40 D$=A$
```

A$, B$ et D$ sont définis à l'intérieur même du programme tandis que C$ est stocké dans l'espace réservé aux chaînes.

Cet espace est déterminé par le premier paramètre de l'instruction CLEAR.

Il existe aussi un espace temporaire pour les opérations sur les chaînes de caractères (Litéral String Pool). Un pointeur vers cette table se trouve dans la région de communication.

## 5.8 La région de communication.

La région de communication peut être divisée en deux grandes parties :

1) La zone de stockage des paramètres et des variables internes. Cette zone contient principalement des éléments de 1 octet (variables internes et sémaphores) et des éléments de 2 octets (variables internes et adresses). Elle commence en 0F380H et se termine en 0FD99H.

2) La table des vecteurs (HOOK) qui permet d'intercepter les principales routines en ROM.

Chaque appel interne à une grande routine du BASIC passe par un vecteur en RAM (donc modifiable). La ROM fait appel à ce vecteur par une instruction CALL.

Chaque vecteur est composé de 3 octets. A l'initialisation, tous les vecteurs sont chargés avec les instructions suivantes :

C9 C9 C9

c'est à dire, une instruction RET (C9) suivie de 2 instructions RET (C9). Ce vecteur renvoie donc directement à l'appeleur sans rien faire.

Pour intercepter une routine, il suffit de remplacer les deux derniers C9 avec l'adresse du programme d'interception et de remplacer le premier RET (C9H) par un JP (C3H).

Le remplacement doit se faire dans l'ordre : d'abord l'adress ensuite le JP. Des exemples d'interruptions seront donnés dans le chapitre 8.

La ROM BASIC étant peu pratique à modifier, cette table de vecteurs est indispensable si on veut modifier des routines internes de la ROM.

Cette table de vecteurs commence à l'adresse FD9AH et se termine en FFC9H.

Vous trouverez toutes les informations sur le contenu de la région de communication à la fin de ce chapitre.



## 5.9 Fonctionnement de la ROM BASIC.

1° Phase : la phase d'entrée.

Elle accepte les entrées en provenance du clavier (rédaction de programme). Après l'impression du message 'Ok', le système est en phase d'entrée.

Fonctionnement succinct de la phase d'entrée.

A) Lire la ligne entrée au clavier.
B) Remplacer les mots clés par leurs codes.
C) Tester si c'est une instruction directe (RUN, CLOAD,...
D) Stocker dans la TIP.

2° Phase : la phase d'interprétation et d'exécution.

Le BASIC MSX est un interpréteur, les lignes sont donc analysées et exécutées les unes après les autres.
Quand on exécute le programme (RUN), le système cherche un code de mot réservé. Une fois ce code trouvé, une adresse est associée à ce code (voir point 5.3) et le contrôle est passé à cette adresse.

Ces différentes adresses sont les points d'entrées des routines de traitement des instructions.
La routine appelée teste la syntaxe de l'instruction (position des virgules, des parenthèses,...).

La phase d'exécution démarre avec une instruction RUN ou GOTO ou lorsqu'une instruction sans numéro de ligne est entrée.

Fonctionnement succinct de la phase d'exécution.

A) Prendre le premier caractère de la ligne courante (TIP)
B) Si le caractère n'est pas un code de mot clé, sauter à
C) Rechercher l'adresse de la routine correspondante.
D) Exécuter la routine en question (test de syntaxe).
E) Retourner à A.
F) Assigner la variable.
G) Evaluer l'expression qui suit la variable.
H) Retourner à A.

La routine d'exécution commence par charger le premier caractère de la ligne courante. Ce caractère est testé ; s'il est supérieur à 80H (128) c'est un code représentant une instruction, le contrôle est alors passé à la routine associée à cette instruction ; s'il est inférieur à 80H, c'est une affectation de variable de la forme X=fonction. La routine d'analyse d'affectation prend le nom de la variable, teste si elle est suivie d'un signe = puis évalue l'expression qui suit le signe =.

Si un code a été trouvé, la routine analyse si ce code est correct car certains codes ne peuvent pas apparaître seuls (THEN, OFF,...) et aucune des fonctions du BASIC (à l'exception de MID$ qui peut apparaître à gauche du signe =) ne peut apparaître seule dans une ligne de BASIC.

Enfin le code est analysé et le contrôle est donné à la routine associée à ce code.

Après chaque routine d'interprétation, un test est effectué pour déterminer s'il y a une marque de fin de ligne ou un symbole ':' de ligne multi-instructions.

Quand on arrive à la fin du programme, le contrôle est automatiquement donné à la routine de traitement de l'instruction END, même si celle-ci est absente.



## 5.10 Fonctions arithmétiques et mathématiques de la ROM MSX

Avant d'analyser les fonctions de la ROM MSX, rappelons les fonctions intrinsèques du processeur Z80.

Le Z80 est capable de réaliser des additions et des soustractions d'entiers de 8 ou de 16 bits. Il ne permet pas la multiplication ou la division.

Ces opérations sont permises entre les registres. Le Z80 ne possède pas d'instructions de calcul entre la mémoire et les registres.

Le BASIC, par contre, supporte les 4 opérations avec des variables de 3 types (entières, simple et double précision)

Ces opérations sont réalisées grâce à des routines internes de la ROM.

A cause de la complexité de ces routines, le mélange de différents types de variables lors d'une opération peut produire des résultats inattendus.

C'est pourquoi vous devez toujours effectuer des opérations entre variables du même type.

Pour contenir une valeur simple ou double précision, les registres du Z80 ne suffisent pas. Une zone tampon dans la zone de communication doit être utilisée.

## 5.11 L'accumulateur virtuel.

Les registres du processeur Z80 ne suffisant pas pour mémoriser une variable simple ou double précision (4 ou 8 octets), une zone de stockage de 8 octets est nécessaire. Cette zone fait partie de la région de communication, elle est située en OF7F6H et est appelée symboliquement ACCUM ou DAC.

Pour les variables en simple précision, les adresses de OF7F6H à F7F9H (4 octets) sont utilisées. Pour les variables en double précision, les adresses de OF7F6H à OF7FDH sont utilisées.

Pour effectuer des opérations entre 2 variables, une deuxième zone de 8 octets est indispensable, elle se trouve à l'adresse OF847H et se nomme symboliquement ACCUM2 ou encore ARG.

Comme ACCUM est utilisé pour tous les types de variables, il est nécessaire de signaler au système le type de variable utilisé. C'est le but du STD (Sémaphore de Type de Donnée), il fait lui aussi partie de la région de communication. Il se trouve à l'adresse (OF663H) et est aussi appelé VALTYP.

Le STD contient un nombre qui indique le type de la variable qui se trouve dans ACCUM ou DAC.

STD = 2 : variable entière
STD = 3 : variable chaîne de caractères
STD = 4 : variable simple précision
STD = 8 : variable double précision

Remarque : le contenu de VALTYP (STD) est en rapport avec la taille utilisée de ACCUM.

Vous trouverez des informations complémentaires sur la structure des variables au chapitre 6, lors de la description de la fonction VARPTR.



## 5.12 Adresses principales de la ROM.

Cette section vous donne les principales adresses de la ROM avec le[s] fonctions. Quand c'est possible, les conditions d'entrée et de sort[ie] données. Cette table n'est ni complète ni exhaustive loin de là. Un[e] description complète de la ROM occuperait à elle seule tout ce manu[el]. Elle fera l'objet d'un autre livre dans cette collection.

Toutes les adresses sont données en hexadécimal.

| Adresse | - - - - F O N C T I O N - - - - |
|---|---|
| 0000 | Point d'entrée de l'initialisation (RESET). |
| 0008 | RST 8 (SYNCHK) :Cette routine regarde le caractère courant par HL et le compare au caractère qui suit le RST, si le caractère n'est pas celui qui est attendu, la routine d'impression de 'SYNTAX ERROR' est appelée, sinon elle sau caractère et revient. Tous les registres sont préservés ex A et HL qui est incrémenté. |
| 0010 | RST 10 (CHRGET) :Cette routine utilise HL comme pointeur e charge dans A le caractère pointé par HL, elle positionne sémaphores du registre F en fonction du type du caractère : si le caractère est numérique, C (carry) est positionné caractère est : (multi-instruction) ou fin de ligne il pos l'indicateur de zéro (Z). Tous les registres sont préservé exceptés A et HL qui est incrémenté. |
| 0018 | RST 18 (OUTDO) :Cette routine sort le caractère contenu da sur le périphérique (CRT ou LPT) déterminé par PRTFLG. Les registres ne sont pas modifiés. |
| 0020 | RST 20 (COMPARE) :Cette routine compare le contenu de HL a contenu de DE ; si HL = DE l'indicateur de zéro (Z) est po tionné, si HL < DE l'indicateur carry (C) est positionné. |
| 0028 | RST 28 : Test du sémaphore de type de donnée (F663H) et po ne les indicateurs du registre F. SIGNE=ENTIER , ZERO=CHAI PARITE=SIMPLE PRECISION, NO CARRY=DOUBLE PRECISION. |
| 0030 | RST 30 : Gestion des SLOTS. |
| 0038 | RST 38 : Gestion des interruptions en provenance du VDP. |
| 003B | Début de la table des JUMPS BIOS (voir section suivante). table se termine à l'adresse 015CH. |
| 01B6 | Routines de gestion des SLOTS mémoires. Ces routines se te en 02D6H. Elle ne sont pas analysées en détail dans ce man |
| 02D7 | Suite de l'initialisation (adresse 0). |

| | |
|---|---|
| 03FB | Processus de traitement du BREAK (CTRL-C). |
| 049D | Initialisation du PSG. |
| 050E | Initialisation mode SCREEN 0. |
| 053B | Initialisation mode SCREEN 1. |
| 05D2 | Initialisation mode SCREEN 2. |
| 061F | Initialisation mode SCREEN 3. |
| 06E4 | Fourni dans HL l'adresse d'un SPRITE dans la TGS si A contient son numéro. |
| 06F9 | Fourni dans HL l'adresse d'un SPRITE dans la TAS si A contient son numéro. |
| 0704 | Fourni dans A le nombre 8 si les SPRITES sont au format 8x8 et 3? si les SPRITES sont au format 16x16. |
| 0744 | Ecriture dans le VDP, DE pointe vers le texte à écrire, BC contient la longueur du texte et HL pointe vers l'adresse de la VIDEORAM où le texte doit être écrit. |
| 07CD | Ecriture dans la VIDEORAM , A contient la valeur à écrire et HL pointe vers l'adresse de la VIDEORAM. |
| 07D7 | Lecture de la VIDEORAM , HL contient l'adresse à lire, au retour A contient la valeur lue. |
| 07DF | Positionne la VIDEORAM à l'adresse contenue dans HL en vue d'une écriture. |
| 07EC | Comme ci-dessus mais en vue d'une lecture. |
| 0815 | Ecriture d'un même caractère plusieurs fois dans la VIDEORAM, HL contient l'adresse de la VIDEORAM , A contient le caractère et BC contient le nombre de fois qu'il faut écrire le caractère. |
| 083B | Retour à l'ancien mode TEXTE (40 ou 32) à la fin d'un programme ou à la suite d'une erreur. |
| 0848 | CLS |
| 084F | Initialise dans le mode SCREEN 0,1,2 ou 3 suivant la valeur de A. |
| 085D | Routine d'impression du contenu de A sur l'imprimante. |
| 0884 | Routine de test de l'état de l'imprimante, si l'imprimante est BUSY, l'indicateur Z est positionné. |
| 088E | Positionnement du curseur en absolu suivant la sequence ESC Y colonne ligne , L contient la colonne et H la ligne. |
| 089D | Traitement du caractère à imprimer pour le rendre compatible avec les imprimantes non MSX. |
| 08BC | Sortie d'un caractère sur le CRT. |
| 092F | Table des valeurs des caractères spéciaux avec leurs adresses de traitement. Cette table se termine en 097FH. |
| 09ED | Affichage du curseur. |
| 0A27 | Effacement du curseur. |
| 0A44 | Déplacement du curseur à droite |
| 0A4C | BACKSPACE |
| 0A57 | Déplacement du curseur vers le haut. |
| 0A5B | Avance du curseur. |
| 0A61 | Déplacement du curseur vers le bas |
| 0A71 | TABULATION |
| 0A7F | HOME |
| 0A85 | Effacement de ligne. |
| 0AB4 | Insertion de ligne. |
| 0AE3 | Effacement du caractère précédent. |



| | |
|---|---|
| 0AFC | Effacement total de la ligne |
| 0AFE | Effacement depuis la position du curseur jusqu'à la fin de ligne. |
| 0B05 | Effacement depuis la position du curseur jusqu'à la fin de page. |
| 0B15 | Effacement des touches de fonction. |
| 0B2B | Affichage des touches de fonction. |
| 0C3C | Routine de traitement de l'interruption hardware générée p VDP. |
| 0D12 | Test de touche clavier enfoncé. |
| 0D6A | Lecture d'un caractère en provenance du clavier. Cette rou réalise une seule scrutation sans attente et sans bouclage |
| 0DA5 | Table de transcodage du clavier. Cette table se termine en |
| 0F06 | Traitement de la touche HOME-CLS. |
| 0F3D | Eteint ou allume le témoin CAPS en fonction du contenu de |
| 0F46 | Traitement de la touche STOP. |
| 0F7A | Positionne le bit 7 du PORT C du PPI en fonction de la val A, ce bit permet des effets sonores. |
| 1021 | Routine de codage de la touche enfoncée. |
| 1033 | Table de codage clavier. Cette table se termine en 10C1H. |
| 10C2 | Mise à jour du pointeur dans le buffer circulaire du clav |
| 10CB | Routine de saisie d'un caractère. |
| 1102 | Routine d'écriture dans le PSG : A contient le numéro du et E contient la valeur à écrire. |
| 110C | Routine de lecture du registre 14 : PORT A du PSG . Au re contient la valeur lue. |
| 110E | Routine de lecture d'un registre du PSG. A l'appel, A con numéro du registre, au retour A contient la valeur lue. |
| 1113 | BEEP : Emission du BEEP. Cette routine détruit tous les r tres. |
| 113B | Routine ACTION : Cette routine lit les informations sur l en cours dans la file musicale (MUSIC QUEUE). En entrée A tient le numéro du canal (0 à 2). Cette routine est utili essentiellement par l'instruction PLAY. |
| 1170 | Routine de positionnement de la fréquence pour l'instruct PLAY. |
| 1181 | Routine de positionnement du volume pour l'instruction PL |
| 1195 | Routine de positionnement de la période d'enveloppe pour truction PLAY. |
| 11EE | Routine de lecture des manettes de jeux ou des flèches du vier. En entrée, A contient 0 pour le clavier, 1 ou 2 pou manette correspondante. |
| 1253 | Routine de lecture des boutons de tir ou de la barre d'es fonction de la valeur de A. |
| 1273 | Routine de lecture des manettes analogiques (PDL). |
| 12AC | Routine de lecture de la tablette analogique (PAD). |
| 1384 | Positionnement du relais de la cassette (MOTOR). |
| 13A9 | Table des valeurs par défaut pour les touches de fonction (F1-F10). Cette table se termine en 1448H |
| 1449 | Lecture du registre d'état du VDP. |

144C Lecture du port A du PPI.

1452 Lecture d'une ligne clavier. En entrée, A contient le numéro de la ligne à lire , en sortie A contient la valeur lue

145F Routine de test de présence de fichier.

146A Comparaison de DE avec HL (RST 20)

1470 Routine de positionnement dans la file musicale. A contient le numéro de la voix.

1474 IDEM 1470 mais le numéro de voix se trouve en FB38H et L doit être positionné en entrée sur la valeur du déplacement

1477 IDEM 1474 mais A doit contenir le numero de voix.

1492 A cette adresse commence la première routine de manipulation des FILES (QUEUES). Les FILES peuvent être de longueur puissance de 2 - 1 et ce jusque 255. Une FILE peut être initialisée à n'importe quel moment et n'importe où. Un pointeur fourni l'adresse de la table des FILES. La table des FILES contient toutes les informations sur chaque FILE. Ces informations sont représentées par 6 octets, le premier donne l'OFFSET pour une mise en FILE, le second donne l'OFFSET pour une prise en FILE, le troisième contient le premier caractère de la FILE, le quatrième la longueur de la FILE et le couple cinquième-sixième, l'adresse de la FILE. Toutes les routines supposent que A contient le numéro de la FILE et que F3F3H contient l'adresse de la table des FILES.

1492 Routine de mise en fin de FILE. Le caractère contenu dans E est mis en FILE, si la FILE est pleine, l'indicateur Z est positionné.

14AD Routine de prise en début de FILE. le caractère est mis dans A, l'indicateur Z est positionné si la file est vide.

14D1 Routine d'écriture du caractère contenu dans E en début de FILE.

14DA Initialise une FILE à vide. B=longueur de la FILE, (DE)=adresse.

14EB Routine qui retourne dans A le nombre d'octets libres dans la FILE.

150F Fin des routines de FILES.

1510 Ecriture d'un caractère dans l'écran en mode graphique. Ce n'est pas une sous-routine.

1599 Routine d'ajustage des valeurs de X et Y . En entrée, BC contient X et DE contient Y. En sortie, ces registres contiennent les mêmes valeurs mais ajustées (MODULO).

15D9 Test de la valeur courante de SCREEN.

15DF Routine de détermination de l'adresse de la VIDEORAM en fonction de la valeur de X et de Y. En entrée, BC contient X et DE contient Y. En sortie HL contient l'adresse de la VIDEORAM et A contient le masque à appliquer. (***** ROUTINE TRES UTILE *****)

160B Table des puissances de 2. Fin en 1612H.

1639 Lecture de l'accumulateur graphique : F92AH contient la localisation dans la VIDEORAM qui est transférée dans HL et F92CH contient le masque qui est transféré dans A.

1640 Ecriture de l'accumulateur graphique (voir ci-dessus).

1647 Lit les attributs de l'accumulateur graphique courant.

1676 Positionne les attributs qui seront utilisés lors des prochaines actions.

167E Positionne le point indiqué par l'accumulateur graphique dans l'octet ATTRBYT.



16AC Les routines suivantes portent sur l'accumulateur graphique défini ci-dessus.

16AC Déplacement d'un point vers la droite avec indicateur C si atteinte d'un bord.

16C5 IDEM 16AC sans indicateur C

16D8 IDEM 16AC mais vers la gauche.

16EE IDEM 16C5 mais vers la gauche.

170A IDEM 16AC mais vers le bas.

172A IDEM 16C5 mais vers le bas.

173C IDEM 16AC mais vers le haut.

175D IDEM 16C5 mais vers le haut.

1809 Routine de remplissage de figure (BOX-FILL)

186C Ecriture d'un PATRON en mode SCREEN 2 . En entrée, A contien PATRON, HL l'adresse dans la table et F3F2H la couleur du PA

18C7 Chargement du coéfficient d'élliptisation pour l'instruction CIRCLE.

18CF Routine utilisée par l'instruction PAINT pour initialiser la couleur des bords.

19DD CASSETTE : Attente puis arrêt du moteur.

19E9 CASSETTE : Arrêt du moteur.

19F1 CASSETTE : Démarrage du moteur puis écriture du HEADER.

1A19 CASSETTE : Ecriture d'un octet.

1A63 CASSETTE : Lecture du HEADER.

1ABC CASSETTE : Lecture d'un octet.

1B46 Routine RST 18 : Ecriture sur l'écran ou sur l'imprimente su l'état de PRTFLG (F416H), le caractère à écrire est contenu A en entrée.

1B63 Sortie du contenu de A sur l'écran.

1BFF Table du générateur de caractères constituée de 256 * 8 octe Fin en 23BEH.

23BF Point d'entrée principal de l'éditeur d'écran (texte BASIC).

23CC Point d'entrée pour la saisie (INPUT) avec production du '?'

23F9 Retour au BASIC.

2439 Table des caractères spéciaux avec l'adresse de traitement. en 2459H

245A Traitement du CR.

24C4 Traitement de CTRL-C.

24E5 Bascule de mode insertion.

24F2 Insertion d'un blanc.

255B Effacement (DELETE) du caractère courant.

2561 Effacement du caractère précédent.

25AE Effacement ligne.

25B9 Effacement fin ligne.

25D7 Ajoute à une ligne existante.

25F8 Positionne sur le mot suivant.

260E Positionne sur le mot précédent.

2624 Déplacement à droite.

2634 Déplacement à gauche.

268C A cette adresse commencent les routines de traitement arithm tiques. (voir commentaires à la section concernée.)

| | |
|---|---|
| 268C | Soustraction double précision : DAC (ACCUM) = DAC - ARG (ACCUM2). |
| 2697 | Addition double précision : DAC = DAC + ARG |
| 26FA | Normalisation d'un résultat. |
| 273C | Routine d'arrondi. |
| 2783 | inversion du signe de DAC. |
| 2797 | SHIFT DAC à gauche d'un chiffre décimal. |
| 27A3 | SHIFT DAC à droite d'un chiffre décimal. |
| 27E6 | Multiplication double précision : DAC = DAC * ARG. |
| 28F9 | Division double précision : FAC = FAC / ARG. |
| 2993 | COSINUS : DAC=COS(DAC) : COS(DAC)=SIN(DAC+PI/2). |
| 29AC | SINUS : DAC=SIN(DAC). |
| 29FB | TANGENTE: DAC=TAN(DAC) : TAN(DAC)=SIN(DAC)/COS(DAC). |
| 2A14 | ARCTANG : DAC=ATN(DAC). |
| 2A72 | LOG : DAC=LOG(DAC). |
| 2AFF | RACINE : DAC=SQR(DAC). |
| 2B4A | EXPON : DAC=EXP(DAC). |
| 2BDF | RANDOM : DAC=RND. |
| 2C88 | Evaluation des polynômes. |
| 2CF1 | Table des constantes pour l'évaluation des fonctions transcendantes (SINUS,COS,TAN,LOG,PI,ATN). Cette table se termine en 2E70H. |
| 2E71 | SIGN : A = SIGN(DAC) |
| 2E7D | ZERO : DAC=0 |
| 2E82 | ABS : DAC=ABS(DAC) |
| 2E86 | NEG : DAC=-DAC |
| 2E97 | SGN : A = SGN(DAC) pour les valeurs entières. |
| 2EB1 | Pousse DAC dans le stack (SP). |
| 2EBE | Pousse un nombre simple précision (SIPR) pointé par HL dans DAC |
| 2EC1 | Pousse le contenu de BC et DE (SIPR) dans DAC. |
| 2ECC | Pousse un nombre simple précision (SIPR) de DAC vers BC et DE dans l'ordre CBED. |
| 2ED6 | Pousse un nombre (SIPR) pointé par HL dans BC et DE dans l'ordre CBED. |
| 2EDF | IDEM dans l'ordre EDCB. |
| 2EE8 | Pousse un nombre (SIPR) de DAC vers la zone pointée par HL. |
| 2EEB | Pousse un nombre (SIPR) de la zone pointée par DE vers la zone pointée par HL. |
| 2EEF | Pousse un nombre pointé par DE vers la zone pointé par HL. VALTYP contient la précision (F663H). |
| 2F05 | IDEM mais de HL vers DAC. |
| 2F0D | IDEM mais de DAC vers HL. |
| 2F21 | Compare 2 nombres simple précision, le premier est dans BC et DE, le second est dans DAC. En sortie, A est 1 si DAC est > , A = -1 si DAC EST < et A=0 si ils sont égaux. |
| 2F4D | Idem mais avec 2 entiers, le premier dans DE et le second dans HL. |
| 2F5C | Idem mais avec 2 nombres double précision, le premier dans ARG (F847H) et le second dans DAC (F7F6H). |
| 2F83 | Idem 2F5C mais avec résultat dans A inversé. |
| 2F8A | Convertit DAC en entier. |



| | |
|---|---|
| 2F99 | Pousse le contenu de HL dans DAC et positionne VALTYP au fo entier. |
| 2FB2 | Force DAC au format simple précision. |
| 2FBA | Convertit un nombre en double précision contenu dans DAC au simple précision. |
| 2FC8 | Convertit un nombre du format entier au format simple préci dans DAC. |
| 303A | Force DAC au format double précision. |
| 3042 | Convertit un nombre contenu dans dac de simple en double précision. |
| 3058 | Force DAC au format chaîne de caractères. |
| 305D | Pousse INT(DAC) dans DE. |
| 30BE | FIX : FIX(DAC)=SGN(DAC)*INT(ABS(DAC)). |
| 30CF | INT : DAC=INT(DAC) |
| 314A | MULTIPLICATION format entier : DE = BC*DE. |
| 3167 | SOUSTRACTION " " : HL = DE-HL. |
| 3172 | ADDITION " " : HL = DE+HL. |
| 3193 | MULTIPLICATION " " : HL = DE*HL. |
| 31E6 | DIVISION " " : HL = DE/HL , reste dans DE |
| 321C | NEGATION " " : HL = -HL. |
| 322B | NEGATION |
| 323A | MODULO " " : HL = DE-DE/HL*HL , DE=quoti |
| 3299 | Chargement d'une constante ASCII dans DAC (ACCUM). Cette r évalue le nombre qui se trouve dans une chaîne de caractèr pointée par HL, la stocke dans DAC et positionne le STD (V en fonction de type de constante. Cette routine s'arrête l elle rencontre une valeur non numérique. Elle accepte les signées exprimées en entier, réel ou en notation scientifi nombre est rendu dans la plus petite petite précision poss |
| 340A | Routine de sortie du message 'in' suivi du numéro de ligne |
| 3425 | Routine de sortie de la valeur contenue dans DAC (ACCUM1) fonction du format indiqué par les registres A,B et C. |
| 3566 | Impression au format simple et double précision. |
| 35A6 | Impression en format fixe suivi de chiffres decimaux. |
| 35EF | Traitement de la notation scientifique 'E'. |
| 3666 | Pousse des 0 dans le BUFFER : HL pointe sur le buffer et A contient le nombre de 0 à pousser. |
| 366E | Pousse des 0 dans le BUFFER avec une virgule ou un point a milieu. En entrée, A contient le nombre de 0, B contient l position du point décimal et C la position de la virgule. pointe sur le BUFFER. |
| 367A | Compte le nombre de virguleset fournit le résultat dans C. |
| 368E | Pousse les virgules et les points décimaux dans le BUFFER. |
| 36B3 | Convertit un nombre simple ou double précision au format dé |
| 36DB | Convertit un entier au format décimal. |
| 371A | Convertit le nombre contenu dans DAC en binaire, octal ou he cimal . |
| 3752 | Prend la longueur et le chiffre le moins significatif de D un nombre réel. |
| 375F | Pousse un espace avant le nombre (pour les positifs). |

377B Routine qui élimine les chiffres à droite du nombre dans DAC (avec arrondi). A contient en entrée le nombre de chiffres à éliminer.
37A2 Compte le nombre de chiffres libres à droite du point décimal.
37B4 Calcule le nombre de chiffres significatifs de la mantisse.
37C8 Exponentiation simple et double précision.
383F Exponentiation entière.
385A Routine d'exponentiation entière proprement dite . Utilise un vieil algorythme indien.
390D HL=HL*DE
391A Positionne l'indicateur de CARRY si ARG peut être converti en entier.
392E Table des adresses des mots clés. Cette table est reprise triée en annexe du présent volume. Fin en 3A3DH.
3A3E Table des adresses pour la recherche des mots clés, une entrée par lettre de l'alphabet (A-Z). Fin en 3A71H.
3A72 Table des mots clés avec le code correspondant (voir annexe). Fin en 3D3AH.
3D3B Table de précédence des opérateurs avec adresse de traitement. Fin en 3D75H.
3D76 Table des messages d'erreur. Chaque message se termine par un octet = 00 . Fin en 3FD1H.
3FD2 Message 'in'
3FD7 Message 'Ok',CR,LF
3FDC Message 'Break'
3FE2 Recherche d'une entrée pour une instruction FOR dans le STACK, pointeur passé dans DE.
4001 Fonction INP
4016 Instruction OUT
401C Instruction WAIT
4039 Traitement de la fin de programme.
411D Point d'entrée principal pour l'impression du message 'Ok' et le retour en mode saisie.
4134 MAIN : Retour au mode saisie.
4295 Fouille le programme pour retrouver la ligne dont le numéro est contenu dans DE. En sortie si l'indicateur de carry n'est pas positionné, la ligne n'est pas trouvée. Si le carry est positionné, alors la ligne est trouvée. L'indicateur de zéro permet de dire si la ligne est supérieure à toutes les lignes déjà existantes ou non.
42B2 Point d'entrée principal du CRUNCHER. Le cruncher sert à convertir tous les mots réservés en CODE, les constantes en format interne et les lignes en format binaire.
4524 Routine de traitement de l'instruction FOR.
4601 Routine d'analyse d'une nouvelle instruction.
4666 CHRGET (RST10H).
4718 Instruction DEFSTR.
471A Instruction DEFINT.
471D Instruction DEFSNG.
4720 Instruction DEFDBL.



4769 Lit un numéro de ligne à la position courante du texte.
479E Instruction RUN.
47B2 Instruction GOSUB.
47E8 Instruction GOTO.
4821 Instruction RETURN.
485E Instruction REM.
486F Instruction IF.
487B Instruction LET.
48E4 Instruction ON GOTO.
495D Instruction RESUME.
49AA Instruction ERRORS.
49B5 Instruction AUTO.
49E5 Traitement de IF THEN ELSE.
4A1D Instruction LPRINT.
4A24 Instruction PRINT.
4B0E Instruction LINE INPUT.
4B9F Instruction READ.
4C5F Routine d'évaluation de formule.
4DC7 Routine EVAL :évaluation d'une expression.
4F57 Traitement des opérateurs relationnels.
4FD5 Fonction USR.
501D DEF FN.
51AD MID$ à gauche.
51C9 Instruction WIDTH.
5229 Instruction LLIST.
522E Instruction LIST.
532E Instruction DELETE.
541C Fonction PEEK.
5423 Instruction POKE.
5468 Instruction RENUM.
558C SYNCHK (RST8)
5597 FSIGN (RST28H)
55A7 Instruction CALL.
579C Début du GENGRP (Routines Graphiques Générales).
579C Scanning d'une coordonnée.
57E5 Instructions PSET et PRESET.
5803 Fonction POINT.
58A7 Instruction LINE.
59C5 Instruction PAINT.
5B11 Instruction CIRCLE.
5D6E Instruction DRAW.
5E94 Instruction DIM.
60B1 Routine de traitement de PRINT USING.
6250 Routine de transfert de blocs.
6275 Traitement de l'erreur OUT OF MEMORY.
6286 Instruction NEW.
62A1 CLEAR : effacement de toutes les variables.
63B1 Routine de TRAPPING ( ON - OFF - STOP ).
63C9 Instruction RESTORE.
63E3 Instruction STOP.

63EA Instruction END.
6424 Instruction CONT.
6438 Instruction TRON et TROFF.
643E Instruction SWAP.
6477 Instruction ERASE.
64AF Instruction CLEAR.
6527 Instruction NEXT.
65C8 Comparaison de 2 chaînes de caractères. En entrée, HL doit pointer vers l'adresse de la première chaîne, BC doit pointer vers l'adresse de la seconde, D doit contenir la longueur de la première chaîne et E la longueur de la seconde. En sortie, le registre A contient FFH, 0 ou 1 suivant le résultat de la comparaison.
65F5 Conversion d'un nombre en OCTAL.
65FA Conversion d'un nombre en HEXA.
65FF Conversion d'un nombre en BINAIRE.
6604 Conversion du nombre contenu dans ACCUM en chaîne de caractères. En entrée, le nombre doit être dans ACCUM, le STD doit être positionné en fonction du type de variable. L'adresse de retour doit être poussée dans la PILE, ensuite poussez HL puis BC, et enfin sautez (JP) à l'adresse 6604H. En sortie, le VARPTR de la chaîne se trouve dans ACCUM et le STD est positionné à 3.
6635 Création d'un espace dans la zone des chaînes de caractères. En entrée, HL+1 pointe sur le premier caractère de la chaîne.
665A Création d'un VARPTR de chaîne de caractères. HL doit pointer sur le premier caractère de la chaîne, la chaîne doit se terminer par un 00H. En sortie, le VARPTR se trouve dans ACCUM avec le STD égal à 3.
668E STRING GARBAGE COLLECTION (Regroupement des espaces).
6787 Concaténation de 2 chaînes de caractères. En entrée, la PILE doit être préparée de la façon suivante: poussez l'adresse de retour, poussez BC puis HL. HL doit contenir l'adresse du VARPTR de la première chaîne et ACCUM doit contenir le VARPTR de la seconde, le STD doit valoir 3. Sautez à l'adresse 6787H (JP). En sortie, le VARPTR de la nouvelle chaîne se trouve dans ACCUM.
67D0 FREE UP STRING TEMPORARY.
67FF Fonction LEN. En entrée, le VARPTR de la chaîne doit être dans ACCUM et le STD doit valoir 3. En sortie, la longueur est un entier dans ACCUM et le STD vaut 2.
680B Fonction ASC. En entrée, le VARPTR de la chaîne se trouve dans ACCUM, STD=3. En sortie, la valeur ASCII du premier caractère se trouve dans ACCUM et STD=2.
681B Fonction CHR$. En entrée ACCUM contient la valeur à convertir. En sortie le VARPTR de la chaîne se trouve dans ACCUM et STD=3.
6861 Fonction LEFT$. En entrée, poussez l'adresse de retour dans la pile, ensuite poussez le VARPTR de la chaîne, chargez B avec le nombre de caractères à extraire et sautez à l'adresse 6861H. En sortie ACCUM contient le VARPTR de la nouvelle chaîne.
6891 Fonction RIGHT$. Fonctionne comme LEFT$.



689A Fonction MID$ à droite. En entrée A contient la position du premier caractère à extraire, et E contient le nombre de ca tères à extraire. Pour le reste voir fonction LEFT$.
68BB Fonction VAL. En entrée le VARPTR de la chaîne se trouve da ACCUM avec STD=3. EN sortie, la valeur se trouve en double précision dans ACCUM avec STD=8.
68EB Fonction INSTR.
69F2 Fonction FRE.
6A0E Ici commencent les routines du disque.
6A0E Lecture du nom de fichier et du DEVICE.
6AB7 Instruction OPEN.
6BA3 Instruction SAVE.
6C14 Instruction CLOSE.
6C2A Instruction FILES et LFILES.
6C35 Instruction PUT et GET.
6C87 Instruction INPUT$
6D03 Fonction LOC.
6D14 Fonction LOF.
6D25 Fonction EOF.
6D39 Fonction FPOS.
6E92 Instruction BSAVE.
6EC6 Instruction BLOAD.
6F15 Analyse du DEVICE ou du DISQUE.
6F76 Table des DEVICES : CAS-LPT-CRT-GRP.
6F8F Traitement du DEVICE.
6FB7 Instruction CSAVE.
703F Instruction CLOAD.
70FF Message 'FOUND'.
7106 Message 'SKIP'.
7328 Envoi d'un CR suivi d'un LF (Routine CRDO).
73B7 MOTOR ON ou OFF.
73C1 Instruction SOUND.
73E4 Instruction PLAY.
7758 Traitement de GET et PUT.
7766 Instruction LOCATE.
77A5 ON STOP
77AB ON SPRITE
77B1 ON INTERVAL
77BF ON STRIG
77D4 ON KEY
77E8 Routine ON KEY.
7838 Routine ON INTERVAL.
786C Instruction KEY ( SET ou LIST ).
7911 Fonction TIME
791B Suite de PLAY
7940 Fonction STICK.
794C Fonction STRIG.
795A Fonction PDL.
7969 Fonction PAD.
7980 Instruction COLOR.

| | |
|---|---|
| 79CC | Instruction SCREEN. |
| 7A48 | Instruction SPRITE. |
| 7A84 | Fonction SPRITE. |
| 7AAF | Instruction PUT SPRITE. |
| 7B37 | Instruction VDP. |
| 7B5A | Instruction BASE. |
| 7BA3 | Table des valeurs par défaut pour BASE. |
| 7BCB | Fonction BASE. |
| 7BE2 | Instruction VPOKE. |
| 7BF5 | Fonction VPEEK. |
| 7C16 | Suite de crochets pour les fonctions et instructions disques non installées. |
| 7C76 | Suite de la routine d'initialisation. |
| 7D31 | Ecriture des messages copyright. |
| 7D5D | Test de la mémoire. |
| 7E4B | Instruction MAX. |
| 7ED8 | Table des messages d'initialisation. |
| 7F27 | Petites routines qui vont s'installer en F380H (gestion des SLOTS). |
| 7F3F | Table des valeurs de la zone de communication. Elles sont copiées dans celle-ci à l'initialisation. |
| 7FDB | Fin de la ROM. |






## 5.13 Table des JUMPS de la ROM BASIC (BIOS).

Cette table vous donne tous les crochets du BIOS. Normalement, tous l
programmes utilisant la ROM doivent passer par ces crochets car le ST
DARD MSX garanti que ces vecteurs ne seront pas modifiés.

| Adresse départ | Saute à adresse | ----------- C O M M E N T A I R E S ----------- |
|---|---|---|
| 0001H | 02D7H | Routine principale d'initialisation (RESET). |
| 0008H | 2683H | RST 8 (SYNCHK) voir contenu de la ROM en 558CH. |
| 000CH | 01B6H | Gestion des slots. |
| 0010H | 2686H | RST 10 (CHRGET) voir contenu de la ROM en 4666H. |
| 0014H | 01D1H | Gestion des slots. |
| 0018H | 1B45H | RST 18 (OUTDO) voir contenu de la ROM en 1B45H. |
| 001CH | 0217H | Gestion des slots. |
| 0020H | 146AH | RST 20 (COMPARE) voir contenu de la ROM en 146AH. |
| 0024H | 025EH | Gestion des slots. |
| 0028H | 2689H | RST 28 (VALTYP) voir contenu de la ROM en 2689H. |
| 0030H | 0205H | RST 30 (SLOTS) voir contenu de la ROM en 0205H. |
| 0038H | 0C3CH | RST 38 (INTVDP) voir contenu de la ROM en 0C3CH. |
| 003BH | 049DH | PSG :Initialisation du PSG. |
| 003EH | 139DH | Initialisation des touches de fonction F1-F10. |
| 0041H | 0577H | VDP :Extinction de l'écran. |
| 0044H | 0570H | VDP :Allumage de l'écran. |
| 0047H | 057FH | VDP :Ecriture dans un registre (B=contenu,C=#reg) |
| 004AH | 07D7H | VDP :Lecture VIDEORAM (HL=adresse -> A=contenu). |
| 004DH | 07CDH | VDP :Ecriture VIDEORAM (HL=adresse,A=valeur). |
| 0050H | 07ECH | VDP :Positionne une adresse en lecture (HL=adress |
| 0053H | 07DFH | VDP :Positionne une adresse en écriture (HL=adres |
| 0056H | 0815H | VDP :Ecriture d'un caractère dans VIDEORAM un cer nombre de fois (HL=adresse,A=valeur,BC=compteur). |
| 0059H | 070FH | VDP :Lecture d'un nombre de caractères dans la VI (HL=adresse,DE=buffer de réception,BC=compteur). |
| 005CH | 0744H | VDP :Ecriture d'un buffer de caractères dans la VIDEORAM (HL=adresse,DE=buffer,BC=compteur). |
| 005FH | 084FH | VDP :Initialisation du VDP en fonction du mode (O (A=mode). |
| 0062H | 07F7H | VDP :Positionne les couleurs de bord et de fond ( contenu de la ROM en 07F7H). |
| 0066H | 1398H | Retour d'interruption avec vecteur crochet en OFD |
| 0069H | 06A8H | VDP :Positionnement des valeurs des tables. |
| 006CH | 050EH | VDP :Positionnement en mode SCREEN 0. |
| 006FH | 0538H | VDP :Positionnement en mode SCREEN 1. |

| | | |
|---|---|---|
| 0072H | 05D2H | VDP :Positionnement en mode SCREEN 2. |
| 0075H | 061FH | VDP :Positionnement en mode SCREEN 3. |
| 0078H | 0594H | VDP :Force mode texte. |
| 007BH | 05B4H | VDP :Force mode graphique 1. |
| 007EH | 0602H | VDP :Force mode graphique 2. |
| 0081H | 0659H | VDP :Force mode multicolore. |
| 0084H | 06E4H | VDP :Si A contient le numéro du SPRITE en entrée, en sortie, HL contient son adresse dans la TGS. |
| 0087H | 06F9 | VDP :Comme ci-dessus, mais HL contient son adresse dans la TAS. |
| 008AH | 0704H | VDP :Fourni dans A la longueur d'un SPRITE (8 ou 32) en fonction du registre R1. |
| 008DH | 1510H | Ecriture d'un caractère en mode graphique, (voir ROM). |
| 0090H | 04BDH | PSG :Initialisation de la file (QUEUE). |
| 0093H | 1102H | PSG :Ecriture dans le PSG (E=valeur,A=numéro registre). |
| 0096H | 110EH | PSG :Lecture d'un registre (A=#REGISTRE->A=contenu). |
| 0099H | 11C4H | PSG :Tâche musicale, (PLAY). |
| 009CH | 0D6AH | KBD :Lecture d'une touche. |
| 009FH | 10CBH | KBD :Attente de pression d'une touche. |
| 00A2H | 08BCH | CRT :Affichage d'un caractère sur écran. |
| 00A5H | 085DH | LPT :Sortie d'un caractère sur imprimante. |
| 00A8H | 0884H | LPT :Test du mot d'état de l'imprimante. |
| 00ABH | 089DH | LPT :Conversion pour caractère non affichable. |
| 00AEH | 23BFH | Point d'entrée principal (PINLIN) pour la saisie. |
| 00B1H | 23D5H | Saisie d'entrée. |
| 00B4H | 23CCH | Affichage d'un '?' et saisie (INPUT). |
| 00B7H | 046FH | CTRL-C ? |
| 00BAH | 03FBH | Processus de traitement du BREAK (CTRL-C). |
| 00BDH | 10F9H | Idem 00BAH. avec HL=0000. |
| 00C0H | 1113H | BEEP. |
| 00C3H | 0848H | CLS. |
| 00C6H | 088EH | Positionnement du curseur. H et L contiennent respectivement la valeur verticale et la valeur horizontale. |
| 00C9H | 0B26H | Affichage des touches de fonction si 0F3DEH # 0 . |
| 00CCH | 0B15H | Effacement des touches de fonction. |
| 00CFH | 0B2BH | Affichage des touches de fonction. |
| 00D2H | 083BH | Retour à l'ancien mode SCREEN à la fin d'un programme ou à la suite d'une erreur. |
| 00D5H | 11EEH | Lecture des manettes de jeux (A=0,1,2)-> A=valeur. |
| 00D8H | 1253H | Lecture des boutons de tir (A=0,1,2)->A=valeur. |
| 00DBH | 12ACH | Lecture de la tablette analogique (PAD). |
| 00DEH | 1273H | Lecture de la palette analogique (PDL). |
| 00E1H | 1A63H | CAS :Lecture du HEADER. |
| 00E4H | 1ABCH | CAS :Lecture d'un octet. |
| 00E7H | 19E9H | CAS :Arrêt du moteur. |
| 00EAH | 19F1H | CAS :Démarrage du moteur puis écriture du HEADER |
| 00EDH | 1A19H | CAS :Ecriture d'un octet. |
| 00F0H | 19DDH | CAS :Attente puis arrêt du moteur. |
| 00F3H | 1384H | CAS : MOTOR ON ou OFF. |
| 00F6H | 14EBH | Retourne le nombre d'octets qui restent dans la file. |
| 00F9H | 1492H | Pose data dans la file (voir ROM). |
| 00FCH | 16C5H | Déplace l'accumulateur graphique d'un point à droite. |



| | | |
|---|---|---|
| 00FFH | 16EEH | " " " " " " gauc |
| 0102H | 175DH | " " " " " en hau |
| 0105H | 173CH | " " " " " " " |
| 0108H | 172AH | " " " " " en bas |
| 010BH | 170AH | " " " " " " " |
| 010EH | 1599H | Ajustage de X et Y (SCALXY) Voir ROM. |
| 0111H | 15DFH | (MAPXY) Voir ROM à l'adresse 15DFH. |
| 0114H | 1639H | Lecture de la valeur de l'accumulateur graphique. |
| 0117H | 1640H | Ecriture " " " " " " |
| 011AH | 1676H | Positionne les attributs de l'accumulateur graphic qui seront utilisés lors des prochaines actions. |
| 011DH | 1647H | Lit les attributs de l'accumulateur graphique cour |
| 0120H | 167EH | Positionne le point indiqué par l'accumulateur gra que dans l'octet ATTRBYT. |
| 0123H | 1809H | Routine de remplissage des rectangles (BOX FILL). |
| 0126H | 18C7H | Chargement du coéfficient d'elliptisation du cercl |
| 0129H | 18CFH | Initialisation de la couleur du bord (PAINT). |
| 012CH | 18E4H | Scrutation des points vers la droite (PAINT). |
| 012FH | 197AH | " " " " " gauche " |
| 0132H | 0F3DH | Eteint ou allume le témoin CAPS. |
| 0135H | 0F7AH | Positionne le BIT 7 du PORT C (SOUND). |
| 0138H | 144CH | PPI :Lecture du port A du PPI (A=contenu). |
| 013BH | 144FH | PPI :Ecriture sur le port A (A=valeur lue). |
| 013EH | 1449H | VDP :Lecture du registre d'état. |
| 0141H | 1452H | PPI :Ecriture du port C et lecture de B (CLAVIER). |
| 0144H | 148AH | Vecteur crochet vers 0FFA7H pour extension. |
| 0147H | 148EH | " " " 0FFACH " " |
| 014AH | 145FH | Test si 0F864H # 0 : présence de fichier. |
| 014DH | 1B63H | (OUTDLP) Sortie du contenu de A sur l'écran. |
| 0150H | 1470H | (GETVCP) A = VOIX (0,1,2) (PLAY). |
| 0153H | 1474H | (GETVC2) L = Déplacement dans le buffer (PLAY). |
| 0156H | 0468H | KBD : Effacement du tampon du clavier. |
| 0159H | 01FFH | Gestion des slots. |

5.14 Principaux paramètres de la région de communication.

| ADR. | NOM. | LG. | - - - - - - F O N C T I O N - - - - - - - - |
|---|---|---|---|
| F380 | SWROM | 11 | Routine de lecture des SLOTS du BANK 0. |
| F385 | | | Routine d'écriture des SLOTS du BANK 0. |
| F38C | JPROM | 14 | Routine de saut à l'interieur d'un SLOT du BANK 0 |
| F39A | USRTAB | 20 | Table des adresses définies par l'instruction DEFUSRn= : occupe 10 x 2 octets. L'adresse de l'USR0 se trouve en F39AH sous la forme classique des adresses du Z80 (partie basse en F39AH, partie haute en F39BH). Avant toute déclaration, chacune des adresses contient un envoi vers le message d'erreur ILLEGAL FUNCTION CALL. |
| F3AE | LLINMO | 1 | Longueur de ligne par défaut en mode SCREEN 0. Vaut 39 en standard. |
| F3AF | LLINM1 | 1 | Longueur de ligne par defaut en mode SCREEN 1. Vaut 31 par défaut. |
| F3B0 | LONLIN | 1 | Longueur de ligne courante. Cette adresse est modifiée par l'instruction WIDTH. |
| F3B1 | LONPAG | 1 | Longueur de page. C'est le nombre de lignes sur un écran. Vaut 24 par défaut. Vous pouvez modifier cette valeur par POKE et changer ainsi la taille de votre écran. |
| F3B2 | TAB | 1 | Nombre de caractères pour TAB. Vaut 14 en standard. |
| F3B3 | TNP0 | 2 | Adresse de la TNP en mode SCREEN 0. |
| F3B5 | TC0 | 2 | " " " TC " " " ". |
| F3B7 | TGP0 | 2 | " " " TGP " " " ". |
| F3B9 | TAS0 | 2 | " " " TAS " " " ". |
| F3BB | TGS0 | 2 | " " " TGS " " " ". |
| F3BD | TNP1 | 2 | " " " TNP " " " 1. |
| F3BF | TC1 | 2 | " " " TC " " " ". |
| F3C1 | TGP1 | 2 | " " " TGP " " " ". |
| F3C3 | TAS1 | 2 | " " " TAS " " " ". |
| F3C5 | TGS1 | 2 | " " " TGS " " " ". |
| F3C7 | TNP2 | 2 | " " " TNP " " " 2. |
| F3C9 | TC2 | 2 | " " " TC " " " ". |
| F3CB | TGP2 | 2 | " " " TGP " " " ". |
| F3CD | TAS2 | 2 | " " " TAS " " " ". |
| F3CF | TGS2 | 2 | " " " TGS " " " ". |
| F3D1 | TNP3 | 2 | " " " TNP " " " 3. |
| F3D3 | TC3 | 2 | " " " TC " " " ". |
| F3D5 | TGP3 | 2 | " " " TGP " " " ". |
| F3D7 | TAS3 | 2 | " " " TAS " " " ". |
| F3D9 | TGS3 | 2 | " " " TGS " " " ". |
| F3DB | KEYCLK | 1 | 0= pas de CLICK sur les touches : 1 = CLICK. |
| F3DC | CSRY | 1 | Position courante verticale du curseur. |
| F3DD | CSRX | 1 | Position courante horizontale du curseur. |
| F3DE | FPFLG | 1 | 0=pas de fonction visible en bas d'écran (F1-F10 1=fonctions visibles. |
| F3DF | VDPR | 8 | Contenu des 8 registres du VDP dans l'ordre 0 à |
| F3E7 | | 1 | vaut 0. |
| F3E8 | | 1 | vaut 255 (0FFH). |
| F3E9 | FORCOL | 1 | Couleur du texte (octet utilisé par COLOR). |
| F3EA | BAKCOL | 1 | Couleur du fond. |
| F3EB | BDRCOL | 1 | Couleur du bord. |
| F3EC | | 3 | Contient C3 00 00 ( JP 0000H ). |
| F3EF | | 3 | Contient C3 00 00 ( JP 0000H ). |
| F3F2 | | 1 | Octet attribut. |
| F3F3 | QUEADR | 2 | Adresse de la table des QUEUES. |
| F3F5 | | 1 | Contient FFH. |
| F3F6 | | 1 | Synchronisation du balayage des touches. |
| F3F7 | | 1 | contient 50H. |
| F3F8 | PUTKBU | 2 | Adresse de l'octet courant à écrire dans le tamp clavier. |
| F3FA | GETKBU | 2 | Adresse de l'octet courant à lire dans le tampon clavier. |
| F3FC | CASATR | 20 | Ces 20 octets constituent les paramètres utiles fonctions de gestion de la cassette. |
| F40F | RSNXTP | | Pointeur pour l'instruction RESUME NEXT. |
| F414 | ERRNUM | 1 | Contient le numéro de la dernière erreur. |
| F415 | LPOS | 1 | Contient la position de la tête de l'imprimante. |
| F416 | LPTFLG | 1 | Sémaphore imprimante : 1 = imprimante : 0 = écra |
| F417 | IMPMSX | 1 | 0=imprimante MSX : 1=imprimante non MSX. |
| F418 | CARFLG | 1 | SI #0 alors le caractère à sortir n'est pas codé |
| F419 | VAL | 2 | Utilisé par la fonction VAL. |
| F41C | CURLIN | 2 | Numéro courant de la ligne en cours d'exécution. |
| F41F | CRUBUF | 316 | Tampon pour le codage d'une ligne BASIC. |
| F55E | KBDBUF | 258 | Zone tampon pour le clavier. |
| F662 | DIMFLG | 1 | Sémaphore de l'instruction DIM. |
| F663 | STD | 1 | Sémaphore qui indique le type de variable préser dans ACCUM (DAC). |
| F664 | OPTYP | 1 | Type d'opérateur. |
| F666 | CURCAR | 2 | Adresse du caractère courant dans le texte. |
| F668 | CODSAV | 1 | Sauvegarde temporaire du code de l'instruction. |
| F672 | MEMSIZ | 2 | Valeur supérieure de la mémoire utilisable par BASIC. Cette valeur est modifiée par l'instructi CLEAR. |
| F674 | STKADR | 2 | Adresse supérieure du SP ( pointeur de pile ). |
| F676 | TXTDEB | 2 | Adresse du début du texte du programme BASIC. |
| F698 | LSPTAD | 2 | Adresse du prochain octet disponible dans la tab des chaînes (LITTERAL STRING POOL TABLE). |
| F69B | STRTOP | 2 | Adresse du sommet de la LSPT. |
| F6A1 | FORPTR | 2 | Pointeur pour l'instruction FOR. |
| F6A3 | LLGRDX | 2 | Adresse de la dernière ligne DATA lue. |
| F6A5 | | 1 | Etiquette pour FOR et USR. |
| F6A6 | | 1 | Etiquette pour INPUT et READ. |
| F6A9 | DIRMOD | 1 | Sémaphore : mode programme ou mode direct. |

| | | | |
|---|---|---|---|
| F6AA | AUTOFL | 1 | Sémaphore : 0=AUTO : 1=PAS AUTO. |
| F6AB | CLN | 2 | Numéro de la ligne courante (utilisé par AUTO). |
| F6AD | ALINC | 2 | Valeur de l'incrément entre 2 lignes (AUTO). |
| F6AF | | 2 | Pointeur pour instruction RESUME. |
| F6B1 | SPADR | 2 | Sauvegarde l'adresse de la PILE pour manipuler une erreur. |
| F6B3 | ERRLIN | 2 | Contient le numéro de la ligne en erreur. |
| F6B5 | CURLIN | 2 | Contient le numéro de la ligne courante. |
| F6B7 | | 2 | Pointeur pour l'instruction RESUME. |
| F6B9 | | 2 | Numéro de la ligne du traitement d'erreur. |
| F6BB | ERRLFG | 1 | Sémaphore : vaut FFH durant l'erreur et 0 après RESUME. |
| F6BE | OLDLIN | 2 | Numéro de ligne aprés STOP ou END. |
| F6C0 | OLDTXT | 2 | Adresse du dernier octet exécuté. |
| F6C2 | VARTAB | 2 | Adresse de la table des variables simples. |
| F6C4 | VTBTAB | 2 | Adresse de la table des variables tableaux. |
| F6C6 | FSLAD | 2 | Adresse du début de l'espace disponible. |
| F6C8 | NCHPTR | 2 | Pointe sur l'octet qui suit le dernier caractère en cours d'exécution. |
| F6CA | VDLT | 26 | Table de déclaration des variables. Composée de 26 octets (1 par lettre de l'alphabet). Chaque octet contient un code qui détermine le type par défaut de chaque variable commençant par cette lettre. En standard, toutes les variables sont définies en double précision (8). |
| F7BC | TEMSWA | 8 | Zone de stockage temporaire pour SWAP. |
| F7C4 | TRCFLG | 1 | Sémaphore : 0 = TROFF : 1 = TRON. |
| F7C5 | BCDBUF | | Début de la zone de travail du progiciel mathématique. |
| F7F6 | ACCUM | 8 | Accumulateur mathématique (encore appelé DAC). |
| F847 | ACCUM2 | 8 | Accumulateur secondaire (encore appelé ARG). |
| F85F | | | Début de la zone des paramètres pour la manipulation des fichiers. |
| F87F | FNCT | | Contenu des touches fonctions (F1-F10). |
| F91F | BASETB | 10 | Valeur courante des tables du VDP. |
| F92A | GENGRP | | Zone de travail pour le progiciel graphique. |
| F931 | CIRCLE | | Zone de travail pour l'instruction CIRCLE. |
| F949 | PAINT | | Zone de travail pour l'instruction PAINT. |
| F956 | PLAY | | Zone de travail pour l'instruction PLAY. |
| F975 | VOICAQ | | Adresse des queues musicales. |
| FBB0 | | 1 | Démarrage à chaud possible si #0. |
| FBB1 | | 1 | #0 si le basic est en ROM. |
| FBCC | CURCOD | 1 | Code du curseur. |
| FBCE | | 10 | Etiquettes pour ON KEY GOSUB. |
| FBD8 | | 1 | Etiquette pour ON ... GOSUB. |
| FBDA | OLDKEY | 11 | Statut de l'ancienne touche. |
| FBE5 | NEWKEY | 11 | Statut de la nouvelle touche. |
| FBF0 | KEYBUF | | Tampon pour le codage de touche. |
| FC48 | BOTTOM | 2 | Adresse du début de la mémoire RAM. |
| FC4A | HIMEM | 2 | Adresse de fin de la mémoire RAM. |
| FC9A | RTYCNT | 1 | Contrôle d'interruption. |
| FC9B | INTFLG | 1 | Sémaphore d'interruption. |



| | | | |
|---|---|---|---|
| FC9C | PADY | 1 | Valeur Y de la manette analogique. |
| FC9D | PADX | 1 | Valeur X de la manette analogique. |
| FCA0 | INTVAL | 2 | Valeur de l'intervalle pou ON INTERVAL GOSUB. |
| FCA2 | INTCNT | 2 | Compteur de l'intervalle. |
| FCA6 | GRPENT | 1 | En tête de caractère graphique. |
| FCA7 | ESCCnt | 1 | Compteur de la séquence ESCAPE. |
| FCA8 | INSFLG | 1 | Sémaphore mode insertion. |
| FCA9 | CSRMOD | 1 | Sémaphore curseur ON ou OFF. |
| FCAA | CURCAR | 1 | Caractère du curseur. |
| FCAB | CAPFLG | 1 | Sémaphore CAPS LOCK. |
| FCAE | BASLOD | 1 | Sémaphore chargement de programme BASIC. |
| FCAF | SCRMOD | 1 | Mode courant de l'écran. |
| FCB0 | OLDMOD | 1 | Ancien mode de l'écran. |
| FCB2 | PANCOL | 1 | Couleur du contour pour PAINT. |
| FCB3 | GCSRX | 2 | Position horizontale du curseur en graphique. |
| FCB5 | GCSRY | 2 | Position verticale du curseur en graphique. |
| FCB7 | GRACX | 2 | Accumulateur graphique X |
| FCB9 | GRACY | 2 | Accumulateur graphique Y |
| FCBB | DRAWFG | 1 | Etiquette pour DRAW. |
| FCBC | SCALE | 1 | Echelle pour DRAW. |
| FCBD | ROT | 1 | Angle pour DRAW. |
| FCBE | | 1 | Sémaphore entrée/sortie binaire. |
| FCC1 | SLOTAR | | Début de la zone de travail pour la commutati cartouche. |
| FD9A | HOOK | | Début de la zone des crochets. |

Cette liste n'est pas complète, mais elle reprend les principaux paramètres de la région de communication.

Les programmes de la section 7 sont remplis d'exemples d'utilisatior ces paramètres.

## 5.15 Table des vecteurs crochets (HOOK).

Voici pour terminer ce chapitre, la table des vecteurs. Chaque entrée se compose de 5 octets. En standard, ces 5 octets ont la valeur C9 (RET). Si on utilise le BASIC DISQUE la plupart de ces vecteurs sont interceptés et ils contiennent alors une instruction C3 (JUMP) suivie d'une adresse de déroutement.

| ADR.HEX | NOM | ----------FONCTION---------- |
|---|---|---|
| FD9A | | Appel en 0C4BH. VDP Traitement des interruptions. |
| FD9F | | Appel en 0C53H. VDP traitement des interruptions, ce vecteur est appelé après la lecture du registre d'état du VDP. |
| FDA4 | CHPUT | Appel lors de l'écriture sur écran du caractère contenu dans A en mode TEXTE |
| FDA9 | | Appel lors de la mise à jour du curseur. |
| FDAE | | Appel lors de l'effacement du curseur. |
| FDB3 | DSPFNK | Appel lors de l'affichage des fonctions F1-F10. |
| FDB8 | ERAFNK | Appel lors de l'effacement des fonctions F1-F10. |
| FDBD | | Appel lors du retour au mode TEXTE (32 ou 40) après un passage en mode graphique 2 ou multicolore. |
| FDC2 | CHRGET | Appel lors de la lecture d'un caractère. |
| FDC7 | | Appel en 071E. Appel lors de l'initialisation du VDP (chargement de la table des caractères...). |
| FDCC | KEYCOD | Appel lors de la lecture clavier au moment ou l'accumulateur contient 10 fois le numéro de la ligne de la touche enfoncée + le numéro de la colonne de cette touche. |
| FDD1 | KEYEAS | Appel en 0F10H. Appel avant de convertir un caractère emis par le clavier d'après la table située en 1003H. |
| FDD6 | NMI | Appel lors du traitement d'une interruption non masquable. |
| FDDB | PINLIN | Appel lors de l'impression de message système. Ce vecteur sert à l'insertion d'une carte 80 colonnes. |
| FDE0 | QINLIN | Appel lors de l'impression d'un ? suivi d'un INPUT. Ce vecteur est intercepté en mode 80 colonnes. |
| FDE5 | INLIN | Appel lors de l'INPUT. Ce vecteur est intercepté en mode 80 colonnes. |
| FDEA | ONGO | Appel lors du traitement des instructions ON GOTO , ON GOSUB . |
| FDEF | DSKO$ | Appel lors de l'instruction DSKO$. Intercepté par SED. |
| FDF4 | SET | Appel lors de l'instruction SET. Intercepté par SED |
| FDF9 | NAME | Appel lors de l'instruction NAME. Intercepté par SED |
| FDFE | KILL | Appel lors de l'instruction KILL. Intercepté par SED |
| FE03 | IPL | Appel lors de l'instruction IPL. Intercepté par SED |
| FE08 | COPY | Appel lors de l'instruction COPY. Intercepté par SED |
| FE0D | CMD | Appel lors de l'instruction CMD. Intercepté par SED |
| FE12 | DSKF | Appel lors de l'instruction DSKF. Intercepté par SED |
| FE17 | DSKI$ | Appel lors de l'instruction DSKI$. Intercepté par SED |
| FE1C | ATTR$ | Appel lors de l'instruction ATTR$. Intercepté par SED |
| FE21 | LSET | Appel lors de l'instruction LSET. Intercepté par SED |
| FE26 | RSET | Appel lors de l'instruction RSET. Intercepté par SED |
| FE2B | FIELD | Appel lors de l'instruction FIELD. Intercepté par SED |
| FE30 | MKI$ | Appel lors de la fonction MKI$. Intercepté par SED |
| FE35 | MKS$ | Appel lors de la fonction MKS$. Intercepté par SED |
| FE3A | MKD$ | Appel lors de la fonction MKD$. Intercepté par SED |
| FE3F | CVI | Appel lors de la fonction CVI. Intercepté par SED |
| FE44 | CVS | Appel lors de la fonction CVS. Intercepté par SED |
| FE49 | CVD | Appel lors de la fonction CVD. Intercepté par SED |
| FE4E | GETPTR | Vecteur intercepté par le SED pour son installation. Utilisé lors du positionnement sur un fichier. |
| FE53 | SETFIL | Appel lors du positionnement d'un pointeur sur un fichier ouvert. |
| FE58 | NOFOR | Appel lors de l'instruction OPEN. Intercepté par SED |
| FE5D | NULOPE | Appel lors de KILL, LOAD, MERGE.. Intercepté par SED |
| FE62 | CLOSE | Appel lors de l'instruction CLOSE. Intercepté par SED |
| FE67 | MERGE | Appel lors de l'instruction MERGE. Intercepté par SED |
| FE6C | SAVED | Appel au début d'une instruction SAVE (SED). |
| FE71 | SAVE | Appel dans le corps d'une instruction SAVE (SED). |
| FE76 | SAVEF | Appel à la fin d'une instruction SAVE (SED). |
| FE7B | FILES | Appel lors de l'instruction FILES. Intercepté par SED |
| FE80 | GETPUT | Appel lors de l'instruction GET ou PUT (SED). |
| FE85 | FILOUT | Appel lors de sortie sur un fichier (SED). |
| FE8A | CHKSEC | Appel lors du test du DEVICE. Permet d'installer d'autres DEVICES. |
| FE8F | INPUT$ | Appel lors de l'instruction INPUT$. |
| FE94 | | Appel lors de la rencontre d'une fonction SED (LOC, L EOF, FPOS). |
| FE99 | LOC | Appel lors de la fonction LOC. Intercepté par le SED |
| FE9E | LOF | Appel lors de la fonction LOF. Intercepté par le SED |
| FEA3 | EOF | Appel lors de la fonction EOF. Intercepté par le SED |
| FEA8 | FPOS | Appel lors de la fonction FPOS. Intercepté par le SED |
| FEAD | | Vecteur utilisé pour interfacer le SED. |
| FEB2 | PARDEV | Appel au début de l'analyse du nom du DEVICE. |
| FEB7 | NODEV | Appel si le nom n'est pas dans la table des DEVICES. |
| FEBC | DEVNAM | Appel si le nom est effectivement celui d'un DEVICE. |
| FEC1 | | CE VECTEUR N'EST PAS UTILISE. |
| FEC6 | GENDSP | Appel lors du traitement d'un DEVICE non DISQUE. |
| FECB | RUNC | Appel lors du NEW ou du RUN. |
| FED0 | CLEARC | Appel lors de l'initialisation de la table des variables. |

| | | |
|---|---|---|
| FED5 | LOPDFT | Appel lors de l'initialisation de la table des variabl (boucle). |
| FEDA | STKERR | Appel lors du nettoyage des FRAME FOR et GOSUB. |
| FEDF | | Appel lors du test de l'existence d'un fichier. |
| FEE4 | OUTDO | Appel lors de la sortie d'un caractère sur écran ou imprimante. |
| FEE9 | CRDO | Appel lors de l'impression d'un CR suivi d'un LF. |
| FEE9 | DEVINP | Appel lors d'INPUT d'un DEVICE. |
| FEF3 | DOGRPH | Appel lors des fonctions graphiques (LINE,CIRCLE,...). |
| FEF8 | PRGEND | Appel à la fin de l'exécution d'un programme. |
| FEFD | | Appel lors de l'impression d'un message d'erreur. |
| FF02 | | Appel à la fin de l'impression du message d'erreur. |
| FF07 | READY | Appel lors de l'impression du message Ok et du retour a mode d'entrée. |
| FF0C | MAIN | Appel à l'entrée de l'interpréteur. |
| FF11 | DIRDO | Appel lors de l'exécution en mode direct. |
| FF16 | FINI | Appel à la fin de l'interprétation d'une instruction. |
| FF1B | FINEND | Appel à la fin de l'interprétation. |
| FF20 | CRUNCH | Appel à l'entrée du CRUNCHER (routine de transformatio d'une ligne BASIC en code de représentation des instructions. |
| FF25 | CRUSH | Appel lors du début de la recheche d'une instruction dans la table alphabétique. |
| FF2A | ISREW | Appel lors de la découverte d'un mot réservé dans la phase de CRUNCH. |
| FF2F | NTFN2 | Appel lorsque le mot réservé est suivi d'un numéro de ligne (GOTO THEN....). |
| FF34 | | Le mot n'est pas réservé. |
| FF39 | SNGFOR | Ce vecteur permet l'installation d'un autre package mathématique. |
| FF3E | NEWSTT | Appel au début d'une nouvelle instruction. |
| FF43 | GONE2 | Appel lors des instructions de déroutement (GOTO,IF..). |
| FF48 | CHRGET | Appel lors de la saisie d'un caractère. |
| FF4D | RETURN | Appel lors du traitement de l'instruction RETURN. |
| FF52 | PRTFLD | Appel lors de l'instruction PRINT. |
| FF57 | COMPRT | Appel dans le corps du traitement de l'instruction PRINT. |
| FF5C | | Appel à la fin du traitement d'une instruction PRINT. |
| FF61 | TRMNOK | Appel lors du traitement d'un DATA ou d'un INPUT incorrect. |
| FF66 | FRMEVL | Appel lors de l'évaluation d'une formule. |
| FF6B | | Permet l'installation d'un autre package mathématique lors de l'évaluation de formule. |
| FF70 | EVAL | Appel lors de l'évaluation d'une expression. |
| FF75 | TRANS | Appel lors de l'évaluation des fonctions transcendantes. Ce vecteur permet l'installation d'un autre package mathématique. |
| FF7A | FINTRA | Appel à la fin de l'évaluation des fonctions transcendantes. |
| FF7F | MID$ | Appel lors du traitement de l'instruction MID$. |



| | | |
|---|---|---|
| FF84 | WIDTH | Appel lors de l'instruction WIDTH. |
| FF89 | LIST | Appel lors de l'instruction (L)LIST. |
| FF8E | BUFLIN | Appel lors de l'instruction LIST au moment de conve le code en MOT CLE. |
| FF93 | POKE | Appel lors de l'instruction POKE. Ce vecteur permet l'installation d'un autre package mathématique. |
| FF98 | SCNEX2 | Appel lors de la conversion d'un numéro de ligne en pointeur et inversément. |
| FF9D | FREEUP | Appel avant la recherche d'une place libre pour une nouvelle chaîne de caractères. |
| FFA2 | | Appel lors de la lecture d'un nom de variable à la position courante dans le texte. |
| FFA7 | | Ce vecteur est utilisé par le CALL BIOS en 145H et pas d'utilité en configuration normale. |
| FFAC | | Ce vecteur est utilisé par le CALL BISO en 148H et pas d'utilité en configuration normale. |
| FFB1 | | Appel lors du traitement d'erreur. |
| FFB6 | LPT | Appel lors de l'impression sur l'imprimante. |
| FFBB | CHKPTR | Appel lors du test du status de l'imprimante. |
| FFC0 | SCREEN | Appel lors de l'instruction SCREEN. |
| FFC5 | PLAY | Appel lors de l'instruction PLAY. |

## 6. LES INSTRUCTIONS MAL CONNUES DU BASIC MICROSOFT.

### 6.1 Généralités.

Dans ce chapitre, nous allons passer en revue les instructions et les fonctions mal connues et donc mal aimées de la plupart des utilisateurs. Ce sont les fonctions en contact direct avec l'assembleur ou le matériel.

CLEAR POKE PEEK OUT INP WAIT
USR DEFUSR VPOKE VPEEK VARPTR
BASE VDP et CALL.

Nous nous attarderons particulièrement sur la fonction USR et les passages d'arguments ainsi que sur la fonction VARPTR qui sera décrite en détail.



### 6.2 Instruction CLEAR.

Syntaxe : CLEAR p1,p2

Le premier paramètre de l'instruction CLEAR, p1, détermine la taille de l'espace réservé aux chaines de caractères (cf 6.7). Si cet espace est trop petit, le message "OUT OF STRING SPACE" se produira. Par défaut, p1 vaut 200.

Le deuxième paramètre, p2, définit l'adresse supérieure au dela de laquelle le BASIC ne peut pas écrire. Ce paramètre est utilisé principalement pour protéger les programmes écrits en langage machine.

Ce paramètre est une adresse la plus proche possible du haut de la mémoire. Par défaut, il vaut F380H pour un système sans disque (début de la région de communi--cation).

Si vous essayez de dépasser F380H, le système affiche le message d'erreur "ILLEGAL FONCTION CALL".

Si vous donnez une adresse trop proche de celle du début de la table des programmes BASIC (TIP), vous n'aurez plus assez de place pour votre programme et le message "OUT OF MEMORY" apparaîtra.

## 6.3 Instruction POKE et fonction PEEK.

### 6.3.1 Instruction POKE.

Syntaxe : POKE adresse, valeur

C'est l'instruction qui permet d'écrire dans la mémoire de façon violente.

L'adresse peut être comprise entre 0 et 65535.

La valeur peut être comprise entre 0 et 255.

EXEMPLE : POKE 50000,87 écrit la valeur 87 dans la mémoire qui se trouve à l'adresse 50000.

Pour un système de base, les adresses comprises entre 0 et 32767 (7FFFH) sont occupées par la ROM BASIC. Il est impossible d'écrire à l'intérieur de celle-ci. Une instruction :

```
POKE 30000,XX
```

serait donc inefficace et sans intérêt.

La zone comprise entre 32768 (8000H) et 62336 (F380H) est réservée au programme BASIC. En faisant des POKE à une adresse comprise dans cette zone, on peut modi-fier un programme BASIC.

EXEMPLE : encoder le programme suivant :

```
10 A$="COUCOU"
```

taper ensuite : POKE &H8006,66

lister le programme. Il est devenu :

```
10 B$="COUCOU"
```

On a remplacé le A (65) par un B (66).

Pour pouvoir écrire dans la mémoire sans problème avec l'instruction POKE, il faut interdire au BASIC de l'uti-liser (à l'aide de l'instruction CLEAR) et ne pas empi-éter sur la région de communication (voir chapitre 6).

Il est parfois intéressant de modifier la région de communication à l'aide de POKE. Vous en trouverez plusieurs exemples dans le chapitre réservé aux trucs et astuces.



### 6.3.2 Fonction PEEK.

Syntaxe : var=PEEK(adresse)

La fonction PEEK permet de lire le contenu d'une mémoire.

Cette fonction ne subit pas les restrictions de l'ins-truction POKE. Vous pouvez donc aller lire le contenu de toutes les mémoires.

La fonction PEEK est très utile pour pouvoir contrôler différents paramètres de la région de communication. Les exemples d'application sont multiples . En voici un choisi parmi tant d'autres :

Pour contrôler le nombre de caractères par ligne (WIDTH courant) faites :

```
PRINT PEEK (-3152)
```

## 6.4 Les instructions VPOKE et VPEEK.

Les instructions VPOKE et VPEEK sont similaires aux instructions POKE et PEEK, mais à la place d'écrire et de lire dans la mémoire centrale, elles écrivent et lisent dans la mémoire du contrôleur d'écran, autrement dit, dans l'écran lui-même.

Une bonne utilisation de ces instructions demande une parfaite connaissance du chapitre 2 reservé au contrôleur d'écran (VDP).

Restriction : les valeurs d'adresses pour VPOKE et VPEEK sont limitées à 16383 (3FFFH).

Les fonctions VPEEK et VPOKE sont transformées en IN et OUT par la ROM BASIC. En effet, on a vu au chapitre 2 que la mémoire du VDP ne fait pas partie de la mémoire centrale, mais est lue par le processeur central (Z80) comme un périphérique connecté à des PORTS entrée/sortie.

Des exemples de VPEEK et VPOKE se trouvent dans le chapitre réservé aux trucs et astuces.

## 6.5 Les Instructions OUT et WAIT et la fonction INP.

### 6.5.1 L'instruction OUT.

Syntaxe : OUT nPORT, valeur

L'instruction OUT permet d'écrire sur un PORT périphérique du Z80.

Le numéro (n) du port doit être compris entre 0 et 255.

La valeur doit être comprise entre 0 et 255.

Tous les PORTS ne sont pas utilisés par le système. L'annexe A du présent volume donne la liste des PORTS utilisés avec leur utilisation.

### 6.5.2 La fonction INP.

Syntaxe : var=INP(nPORT)

Cette fonction lit le contenu d'un port du Z80.

Le numéro du port doit être compris entre 0 et 255.

La remarque faite pour l'instruction OUT est valable pour la fonction INP, seuls quelques PORTS ont intérêt à être lus.

### 6.5.3 L'instruction WAIT.

Syntaxe : WAIT nPORT,octet de masque, octet de sélection

Cette instruction lit le contenu du port numéro nPORT, lui applique une fonction ET LOGIQUE avec l'octet du masque, puis une fonction OU EXCLUSIF avec l'octet de sélection et ne rend la main au programme que lorsque le résultat est différent de 0.



La fonction de masque permet d'isoler le ou les bit(s) à tester.

La fonction de sélection permet d'inverser l'état à tester.

EXEMPLE : je désire attendre tant que le bit le plus significatif (B7) du port 133 est 0.

j'écris : WAIT 133,&B10000000

ou encore : WAIT 133,128

si je désire l'inverse, c'est à dire attendre tant que le bit 7 du port 133 est différent de 0, j'écris :

WAIT 133,&B10000000,&B10000000

ou encore : WAIT 133,128,128





## 6.6 L'instruction DEFUSR et la fonction USR.

Le BASIC est un langage facile à utiliser et très efficace pour les calculs mathématiques et les programmes de gestion.

Mais, lorsqu'une exécution ultra-rapide ou une économie de mémoire est nécessaire, on doit s'adresser au processeur dans sa langue maternelle : le langage machine.

A l'exception des jeux d'arcades, il est rarement pratique d'écrire un programme complet en langage machine, cette écriture étant fastidieuse et longue.

La meilleure approche consiste à réaliser le programme en BASIC et à programmer les sous-routines trop longues en BASIC ne langage machine.

L'utilisation de sous-programmes en langage machine dans un programme BASIC demande quelques précautions.

1) Interdire au programme BASIC et à ses tables de rentrer en conflit avec lui au point de vue de l'emplacement. Ceci est résolu grâce à l'instruc--tion CLEAR.

2) Introduire (charger) le programme. Nous analyserons dans la suite de ce chapitre les différentes façons de procéder.

3) Définir le point d'entrée. C'est le but de l'instruc--tion DEFUSR.

Syntaxe : DEFUSRn=adresse

où n est compris entre 0 et 9.

On peut donc définir 10 USR simultanément.

4) Exécuter la routine. C'est le but de la fonction USR.

Syntaxe : X=USRn(Y)

où n est compris entre 0 et 9.

Ainsi, l'exécution sera lancée à l'adresse définie par le DEFUSR correspondant.



X=USR5(Y) lance l'exécution à l'adresse définie par le DEFUSR5.

REMARQUE : si l'adresse de départ n'a pas été définie, le système peut se "planter" ou se réinitialiser.

X est la variable qui peut recevoir un résultat de la routine, Y est une variable entière ou un octet que l'on peut transmettre à la routine.

5) Passer des valeurs du BASIC au langage machine.

On peut passer un argument (ou une variable) du BASIC vers le langage machine. Deux cas peuvent se présenter :

A- passer un nombre entre 0 et 255.
Il suffit de faire : CALL 521FH (CD 1F 52) comme première instruction de la sous-routine en langage machine et on récupère l'argument dans l'accumulateur

B- Passer un nombre entier sur deux octets, ou sur une adresse, entre 0 et 65535. Il suffit de faire: CALL 2F8AH (CD 8A 2F) et on récupère l'argument dans HL.

6) Passer une valeur au BASIC et y retourner.

Pour retourner au BASIC sans lui passer de valeur, il suffit de faire RET (C9).

Pour retourner au BASIC en lui transmettant un entier entre 0 et 65535, il suffit de mettre ce nombre dans HL et de faire : JP 2F99H (C3 99 2F)

Lors du retour au BASIC, le contenu de HL se trouve dans la variable qui est devant le signe = de la fonction USR.

## 6.7 Introduire ou charger un programme en langage machine.

### 6.7.1 Méthode par DATA et POKE.

C'est la méthode la plus simple. Elle est utilisée par tous les exemples de ce manuel. Ce n'est pas la meilleure ni la plus originale.

Procédure : on met les valeurs à introduire en mémoire (en décimal ou en hexadécimal) dans des lignes de DATA, ensuite on les lit et on les envoie en mémoire par des POKE et ce, au moyen de la boucle FOR...NEXT.

EXEMPLE : écrire les 5 octets 00,00,00,00,221 à l'adresse 50000.

```
 10 FOR I=50000 TO 50004
 20 READ A
 30 POKE I,A
 40 NEXT I
100 DATA 00,00,00,00,221
```

Autre exemple où l'on travaille en hexadé--cimal : écrire les 5 octets 3E,20,00,00,C9 à l'adresse D000H.

```
 10 FOR I=&HD000 TO &HD004
 20 READ A$
 30 POKE I,VAL("&H"+A$)
 40 NEXT I
100 DATA 3E,20,00,00,C9
```

### 6.7.2 Méthode de la chaîne de caractères.

On peut charger n'importe quelle routine à l'intérieur d'une chaîne de caractères, à condition qu'elle soit inférieure à 255 octets.

Cette méthode présente plusieurs avantages :

Le Clear n'est pas nécessaire.
Le stockage est aisé.
Le déplacement est facile.
Pas d'attente pour le chargement.



Et plusieurs inconvénients :

Limité à 255 caractères.
Les octets égaux à 00 sont gênants.
Le programme doit être indépendant de l'adresse d'implantation.

Description de la méthode :

1) construire au début du programme une chaîne de caractères de la longueur du programme à insérer.

2) installer la routine de construction en fin de programme.

3) exécuter le programme de construction.

4) effacer la routine de construction et garder la ligne contenant la chaîne construite.

5) ecrire le programme qui va utiliser la routine en langage machine, le point d'entrée de la routine étant déterminé par la formule suivante:

```
AD=PEEK(VARPTR(A$ )+1)+256 * PEEK(VARPTR(A$)+2)
```

en supposant que la chaîne de caractères du 1) a été baptisée A$.

EXEMPLE : On installe une routine de 5 octets dans la variable ZZ$

```
   10 ZZ$="*****" ; REM   LONGUEUR 5
50000 AD=PEEK(VARPTR(ZZ$+1)+256*PEEK(VARPTR(ZZ$
50010 LG=5
50020 FOR I=1 TO LG
50030 READ X$
50040 AD+I-1,VAL("&H"+X$)
50050 NEXT I
50060 DATA 3E,20,3E,40,C9
```

### 6.7.3 La méthode de la variable tableau.

On peut aussi charger une routine en langage machine dans une variable tableau entière.

Cette méthode présente les avantages suivants :

1) Le CLEAR n'est pas nécessaire.
2) Le transfert d'argument est très facile.
3) On peut utiliser des octets égaux à 00.

Et l'inconvénient suivant :

Le programme doit être indépendant de l'adresse.

Description de la méthode :

1) Définir la variable tableau comme variable entière.
2) Diviser le nombre d'octets du programme par 2 et prendre le plus grand entier - 1.
3) Dimensionner la variable avec la valeur ainsi trouvée.
4) Calculer la valeur de chaque élément du tableau en utilisant la formule suivante :

   X= octet(n) + 256 * octet(n+1)
5) Etablir les égalités d'éléments.
6) Définir le point d'entrée. Il est égal au VARPTR de l'élément 0 de la variable tableau.

EXEMPLE : Soit le programme de 5 octets suivants :
3E, 10, 3E, 40, C9

```
10 DEFINT A     :  REM variable A entière
20 DIM A(2)     :  REM 2=INT(5/2+1)-1
30 A(0)=4158    :  REM 4158= 16(10H) * 256 + 62(3EH)
40 A(1)=16446   :  REM 16446= 64(40H) * 256 + 62(3EH)
50 A(2)=201     :  REM 201= 0 * 256 + 201(C9H)
60 DEFUSR0=VARPTR(A(0))
```



REMARQUE : Si une valeur est supérieure à 32767, il faut lui soustraire 65536.

Exemple : 3ED2
A(n)= 210(D2H) *256 + 62(3EH) = 53822.
53822 > 32767 donc A(n)=53822-65536=-11714

D'autres possibilités existent. On peut stocker un programme en langage machine dans une ligne de REMARQUE, avant le début d'un programme BASIC en modifiant les pointeurs,...

## 6.8 La fonction VARPTR.

La fonction VARPTR est un des plus merveilleux outils du BASIC MICROSOFT. Elle permet d'atteindre l'adresse de stockage des valeurs assignées aux variables ainsi que différentes informations sur leur contenu.

A l'aide des adresses obtenues par la fonction VARPTR, de l'instruction POKE et de la fonction PEEK, on peut effectuer une foule d'opérations très utiles.

L'utilisation principale de la fonction VARPTR est certainement de retrouver des informations sur les chaînes de caractères.

Lorsqu'on écrit : 10 A$="COUCOU", le système d'exploitation de l'interpréteur BASIC doit sauvegarder la valeur affectée à A$ (en l'occurence COUCOU) quelque part dans la mémoire (voir l'espace réservé aux chaînes dans le chapitre 5).

Lorsque, quelques lignes plus bas, on écrit : 50 PRINT A$, le système devra être capable de retrouver le COUCOU.

Pour effectuer cette opération, le BASIC possède une liste des variables utilisées. Chaque fois qu'il rencontre une nouvelle variable, il l'ajoute à cette liste.

La variable qui a été rencontrée la première dans le programme sera la première dans la liste, et celle qui sera rencontrée la dernière dans le programme, sera la dernière dans la liste.

Chaque fois que le BASIC rencontre une nouvelle variable, il fouille la liste pour voir si cette variable a déjà été affectée. Si ce n'est pas le cas, il l'ajoute à la liste.

Le BASIC possède deux listes : une pour les variables simples, et une pour les variables dimensionnées. Le système consulte la liste appropriée à la variable rencontrée.

Remarque : le temps pris par le système pour retrouver une variable est un facteur influençant très fort la vitesse d'exécution des programmes. Il est possible d'améliorer de façon notable la vitesse d'exécution d'un programme en définissant au début du programme une liste des variables le plus souvent utilisées.



Les variables simples sont définies la première fois qu'on leur attribue une valeur, les variables tableau sont définies lors de l'instruction DIM.

En plus du nom de la variable, la liste contient des informations sur le type de variable (entière, simple précision, double précision, chaîne).

En fonction de ce type, d'autres informations sont fournies au système : soit la valeur directe de la variable, soit l'adresse où l'on peut retrouver cette valeur.

Le BASIC utilise ces informations pour retrouver rapidement les valeurs lors de l'exécution d'un programme.

Toutes ces informations sont aisées à accéder grâce à l'instruction VARPTR.

L'instruction X=VARPTR(A$) fournira une valeur X, adresse où des informations sur A$ pourront être trouvées.

La variable sur laquelle on demande des renseignements peut très bien être une variable entière, simple ou double précisión, une variable tableau ou même une variable tableau de chaîne.
X=VARPTR(A$(2)) est parfaitement valable.

L'utilisation qu'on peut faire de l'adresse contenue dans X est fonction du type de variable.

La valeur contenue dans X étant une adresse, elle est comprise entre 0 et 65535. Donc, c'est une valeur entière et elle tient sur deux octets.

Le contenu de l'adresse fournie par la fonction VARPTR varie en fonction du type de variable.

Si AD est l'adresse fournie par la fonction VARPTR et si la variable est :

1° Une variable simple :

A) Variable entière, 2 octets.
   Une variable entière est stockée sur 16 bits en binaire signé. Le bit le plus significatif (B15) étant le bit de signe.
   AD contient les 8 bits les moins significatifs de la variable.
   AD+1 contient les 8 bits les plus significatifs de la variable.

EXEMPLES :

Si la variable vaut 12345, AD contient 57 (39H) et AD+1 contient 48 (30H).
Car : 48 x 256 + 57 = 12345.

Si la variable vaut -12345, AD contient 199 (C7H) et AD+1 contient 207 (CFH).
Car : 207 x 256 + 199 = 53191 et 53191 - 65536 = -12345.

B) Variable simple précision, 4 octets.
AD contient le signe de la mantisse (B7=0 si positif et 1 si négatif), le signe de l'exposant (B6=0 si négatif et 1 si positif) et la valeur de l'exposant de la variable sur 6 bits (B5-B0) en mode signé.
L'exposant peut donc être compris entre +63 et -63.
De AD+1 à AD+3, la mantisse de la variable (6 chiffres significatifs), nombre inférieur à 1, est stockée en BCD (Binaire Codé Décimal).

EXEMPLES :

Si la variable vaut 12345 (0,12345 X 10 exp 5).
AD contient :
signe positif : Bit 7 = 0
exposant positif = Bit 6 = 1
Valeur de l'exposant = 5 : B5 à B0 = 000101
AD contient donc : 01000101 c.à.d. 69 (45H)
AD+1 contient 18 (12H)
AD+2 contient 52 (34H)
AD+3 contient 80 (50H)

Si la variable vaut -735,4 (-0,7354 X 10 exp 3).
AD contient :
signe négatif : Bit 7 = 1
exposant positif : Bit 6 = 1
Valeur de l'exposant = 3 : B5 à B0 = 000011
AD contient donc 11000011 c.à.d. 195 (C3)
AD+1 contient 115 (73H)
AD+2 contient 84 (54H)
AD+3 contient 0 (00H)

Si la variable vaut -0,000654 (-0,654 X 10 exp -3).
AD contient :
signe négatif : Bit 7 = 1
exposant négatif : Bit 6 = 0
Valeur de l'exposant : -3 : B5 à B0 = 111101 (binaire signé)
AD contient donc 10111101 c.à.d. 189 (BD)
AD+1 contient 101 (65H)
AD+2 contient 64 (40H)
AD+3 contient 0 (00H)



C) Variable double précision, 8 octets.
Le système de stockage de ces variables est identique à celui des variables double précision, à l'exception de la mantisse qui peut contenir 14 chiffres significatifs codés en BCD sur 7 octets (AD+1, AD+7).

D) Variable chaîne de caractères, 3 octets.
AD contient la longueur de la chaîne.
AD+1 contient la valeur basse de l'adresse à laquelle on trouve le contenu de la variable. Cette adresse correspond à une zone de la Table des Instructions de Programme ou à une zone de l'Espace Réservé aux Chaînes (voir ch.5).
AD+2 contient la valeur haute de cette adresse.

E) Quel que soit le type de variable simple, AD-1 contient le deuxième caractère du nom de la variable ou 00 si le nom ne comporte qu'une lettre. AD-2 contient la première lettre du nom de la variable et AD-3 contient le code du type de la variable (2=entier, 4=simple précision, 8=double précision, 3=chaîne de caractères).

2° Une variable dimensionnée.

Les points A à D sont identiques, que la variable soit simple ou dimensionnée. Le point E est totalement différent.

Les variables dimensionnées sont déclarées en même temps par l'instruction DIM, elles ont donc des points d'entrée jointifs et consécutifs dans la Table des Variables (TV).

AD-1 du VARPTR de la variable d'indice 5, par exemple, est donc le dernier octet de la variable d'indice 4, et ainsi de suite.

Pour obtenir les renseignements équivalents au point E de la Table des Variables Simples, il faut que AD soit l'adress fournie par la fonction VARPTR de l'élément d'indice 0.

Pour le VARPTR de l'élément d'indice 0, AD-1 et AD-2 contiennent la dimension de la variable augmentée de 1 (autrement dit, le nombre d'éléments de la variable AD-3 contient le nombre de dimensions de la variable (nombre d'indices).
AD-4 et AD-5 contiennent le nombre d'octets à ajouter à AD-4 pour arriver au début de la variable suivante (OFFSET).
AD-6 contient le deuxième caractère du nom de la variable ou 00 si le nom ne comporte qu'un caractère. AD-7 contient la première lettre du nom de la variable. AD-8 contient le typ de la variable.

Exemple : si la variable CX est dimensionnée à 5, est définie comme entière, et si AD=VARPTR(CX(0)).

```
AD-8 contient 4     :  Type simple précision.
AD-7 contient 43H   :  Valeur ASCII de la lettre 'C'.
AD-6 contient 58h   :  Valeur ASCII de la lettre 'X'.
AD-5 contient 0FH   :  Valeur de l'offset = 256 * la
AD-4 contient 00H   :  valeur de AD-4 + valeur de AD-5
AD-3 contient 01H   :  variable à une dimension
AD-2 contient 06H   :  256 * la valeur de AD-1 + valeur de
AD-1 contient 00H   :  AD-2=6 =nombre d'éléments = DIM+1.
```

Remarque :

Lors de vos essais avec les VARPTR des variables dimensionnées, définissez toutes vos variables simples avant vos variables dimensionnées, car chaque nouvelle définition d'une variable simple modifie la position du début de la Table des Variables Dimensionnées et par là, la position des VARPTR de ces variables.



## 6.9 Les fonctions définies par l'utilisateur (DEFFN).

Je suppose que comme des milliers d'utilisateurs du BASIC MICROSOFT, vous n'utilisez jamais les fonctions définies par l'utilisateur. Utiliser de telles fonctions n'apparait pas immédiatement nécessaire au programmeur débutant et les exemples des manuels ne montrent pas leur utilité.

Pourtant, ces fonctions permettent des techniques de programmation particulièrement intéressantes.

* Avantages :

Les variables utilisées dans la fonction ne sont pas affectées par un appel.

Les fonctions peuvent être définies n'importe où dans le programme, à condition que la logique du programme rencontre la définition au moins une fois avant un appel.

On peut redéfinir une fonction autant de fois que nécessaire.

On peut définir une fonction qui utilise d'autres fonctions définies.

Une définition de fonction peut appeler un USR.

* Inconvénient :

Une fonction définie ne peut pas contenir d'instruction.

SYNTAXE :

Déclaration : DEFFN NN(P1,P2,...,Pn) = fonction BASIC.

Utilisation et appel : FN NN(P1,P2,...,Pn).

Exemples :

1° On veut réaliser une fonction qui donne la valeur hexadécimale d'une adresse mémorisée sous la forme classique (2 octets) à une autre adresse X.

On peut écrire : AD$ = HEX$(PEEK(X)+256*PEEK(X+1))

On peut aussi écrire :

DEFFN AD$=HEX$(PEEK(X)+256*PEEK(X+1))

et chaque fois qu'on devra faire un appel à la fonction, il suffira d'écrire : FN AD$(Z) où Z est soit la valeur de l'adresse qui contient l'adresse à rechercher, soit une variable qui contient cette valeur.

2° On veut réaliser une fonction qui centre une chaîne de caractères dans un espace de N caractères.

On peut écrire :

DEFFN CT$(A$,N%)=STRING$(N%/2-LEN(A$)/2-.5," ")+A$

L'appel s'effectuant grâce à la fonction : FNCT$(Z$,I%) où Z$ est la variable à centrer et I% le nombre de caractères de la ligne.

D'autres exemples d'utilisation des fonctions définies par l'utilisateur sont données au chapitre réservé aux trucs et astuces.



## 6.10 Instructions BASE et VDP.

### 6.10.1 Instruction BASE.

L'instruction BASE est comme l'instruction VDP à la fois une instruction et une fonction.

En format instruction, BASE a la syntaxe suivante :

BASE(n) = expression

où n est un entier compris entre 0 et 19. Il y a donc 20 possibilités pour cette instruction.

L'instruction BASE sert à positionner une adresse pour une table dans un mode déterminé du VDP (Vidéo Display Processor)

A chaque nombre (0-19) correspond une valeur d'adresse pour une des tables pour un mode (5 tables et 4 modes font bien 20 possibilités).

A la valeur 0 correspond l'adresse de la TNP pour le mode texte.

| valeur | mode | table | valeur | mode | table |
|---|---|---|---|---|---|
| 0 | TEXTE | TNP | 10 | GRAPHIQUE II | TNP |
| 1 | TEXTE | TC | 11 | GRAPHIQUE II | TC |
| 2 | TEXTE | TGP | 12 | GRAPHIQUE II | TGP |
| 3 | TEXTE | TAS | 13 | GRAPHIQUE II | TAS |
| 4 | TEXTE | TGS | 14 | GRAPHIQUE II | TGS |
| 5 | GRAPHIQUE I | TNP | 15 | MULTICOLORE | TNP |
| 6 | GRAPHIQUE I | TC | 16 | MULTICOLORE | TC |
| 7 | GRAPHIQUE I | TGP | 17 | MULTICOLORE | TGP |
| 8 | GRAPHIQUE I | TAS | 18 | MULTICOLORE | TAS |
| 9 | GRAPHIQUE I | TGS | 19 | MULTICOLORE | TGS |

L'expression à droite du signe égal peut prendre une valeur quelconque parmi les adresses de la VIDEORAM. ( 0 à 16384 ).

EXEMPLE : positionnement de la TNP en mode texte à l'adresse 0800H.

BASE(0)=&H800

La fonction BASE a comme syntaxe : var=BASE(n).

Elle permet simplement de lire la valeur de l'adresse attribuée à une Table dans un mode donné.

EXEMPLE : C=BASE(12) mettra dans C un entier égal à l'adresse de la RGP en mode graphique II.

REMARQUE : Les valeurs de BASE sont stockées dans la région de communication à l'adresse

OF3B3H + 2 X n

où n est le numéro de la base à trouver.

### 6.10.2 Instruction VDP.

L'instruction VDP, comme l'instruction BASE, a un format instruction de la forme :

VDP(n) = expression

et un format fonction de la forme :

VAR = VDP(n)

Cette instruction permet de positionner directement la valeur d'un des registres du VDP ou de lire le contenu de celui-ci.

n peut prendre une valeur entre 0 et 8 et exprimer une valeur entière entre 0 et 255 (registre de 8 bits).

La bonne utilisation des instructions BASE et VDP prouve une parfaite compréhension du chapitre II.



### 6.11 Instruction CALL.

L'instruction CALL permet d'ajouter des mots clés au BASIC .

Syntaxe : CALL nom de routine (liste d'arguments)

Le BASIC, lorsqu'il rencontre une instruction CALL, essaye de retrouver le nom de la routine parmi les ROM qui lui sont connectées sur les différents SLOTS.

Le BASIC se charge uniquement de la reconnaissance du nom de la routine, la liste d'arguments optionnelle doit être analysée par la ROM contenant la routine.

Si aucune ROM ne reconnait le mot clé, un message SYNTAX ERROR se produit.

La programmation de fonctions CALL nécessitant la fabrication d'EPROMS, nous n'analyserons pas plus en détail cette fonction dans le présent volume.



# 7. TRUCS, ASTUCES ET PROGRAMMES.

## 7.1 Généralités.

Dans ce dernier chapitre, vous trouverez des petits trucs et astuces qui utilisent les connaissances acquises à la lecture des chapitres précédents, autrement dit, la connais--sance des sous-routines internes de la ROM BASIC, des vecteurs, des variables de la région de communication et des périphériques (VDP,PSG et PPI).

Vous y trouverez aussi divers programmes plus conséquents qui vous permettront entre autres de transformer votre clavier QWERTY en AZERTY, d'introduire des fonctions cal--culables dans une instruction INPUT et de construire auto--matiquement des lignes de DATA avec des données de la mémoire centrale.

Enfin, vous trouverez des exemples de fonctions définies par l'utilisateur, un exemple sur l'utilisation des varia--bles tableaux pour le stockage de programmes en langage machine et surtout, un moniteur et un générateur de caractères.



## 7.2 Trucs et astuces.

### 7.2.1 Réservation de mémoire avant le début de la TIP.

La région de communication contient l'adresse de départ de la TIP (Table des Instructions de Programme) augmentée de 1 à l'adresse 0F676H et 0F677H.

Il suffit de modifier cette valeur par deux POKES judicieux, de mettre un 00 à la nouvelle adresse et d'effectuer un NEW.

Exemple: positionner la TIP en 9000H.

```
10 POKE &HF676,1
20 POKE &HF677,&H90
30 POKE &H9000,0
40 NEW
```

A partir de cet instant, les adresses de 8000H à 9000H ne sont plus touchées par le BASIC. Vous pouvez y installer des routines en langage machine.

### 7.2.2 Scrutation du BUFFER clavier.

Ce petit programme permet de scruter le clavier sans faire d'INKEY$ ou d'INPUT et il détecte toutes les touches.

Il utilise le tampon (BUFFER) de la région de communication. Il suffira de noter la valeur qui correspond à la touche que l'on veut détecter et de faire les tests en fonction de cette valeur

```
10 FOR I=&HFBE5 TO &HFBEF
20 PRINT PEEK(I);" ";
30 NEXT I
40 PRINT
50 GOTO 10
```

Lancez le programme, appuyez sur les touches et notez les val

7.2.3 Modification du message 'Ok'

Pour remplacer le message 'Ok' par un autre message (PRET par exemple), il suffit d'intercepter le vecteur d'affichage à l'adresse 0FF07H.

```
 10 REM Ce programme change le mot de sollicitation 'Ok'
 20 CLEAR 200,&HF000
 30 AD=&HFF07      ; REM vecteur crochet
 40 POKE AD+1,0    ; REM partie basse de l'adresse
 50 POKE AD+2,&HF0 ; REM partie haute de l'adresse
 60 FOR I=&HF000 TO &HF011
 70 READ A$
 80 POKE I,VAL("&H"+A$)
 90 NEXT I
 95 POKE AD,&HC3
100 DATA CD,23,73,21,09,F0,C3,31,41,0A,0D,50,52,45,54,0D,0
110 DATA 00
```

Programme assembleur contenu dans le programme BASIC :

```
F000  CD 23 73    DEBUT   CALL    07323H    Faire comme RO
F003  21 09 F0            LD      HL,MSG    HL pointe sur M
F006  C3 31 41            JP      4131H     Suite programm
F009  0A          MSG     DEFB    0AH       LINE FEED
F00A  0D                  DEFB    0DH       RETOUR CHARIOT
F00B  50 52 45 54         DEFM    'PRET'
F00F  0D                  DEFB    0DH
F010  0A                  DEFB    0AH
F011  00                  DEFB    00H       Fin du message
```

7.2.4 Suppression de l'instruction LIST.

Pour supprimer l'instruction LIST dans un but de protection, il suffit d'intercepter le vecteur en 0FF89H et de remplacer le RETURN (C9) par un saut (JP) à l'adresse d'introduction de ligne BASIC (411DH).

```
POKE &HFF8A,&H1D : POKE &HFF8B,&H41 : POKE &HFF89,&HC3
```



7.2.5 Modification des messages d'erreur.

Ce petit programme est réservé aux possesseurs de systèmes équipés de 64K de mémoire RAM.

Pour modifier les messages d'erreur, vous devez au préalable copier la ROM dans la RAM 32K des 2 BANKS inférieurs à l'aide du programme de la section 7.9.

Les messages d'erreur se trouvent dans la ROM de l'adresse 03D76H à l'adresse 03FD1H dans l'ordre des numéros d'erreur. Vous avez donc à votre disposition 603 octets pour traduire les messages.

Un message doit se terminer par 0.

Voici un programme BASIC qui modifie les 3 premiers messages. Il vous reste à le compléter sans dépasser les 603 octets.

```
 10 REM Avez vous lancé  le programme de copie de la ROM
 20 AD=&H3D76
 30 FOR I=1 TO 3: REM modifiez ce nombre en fonction du
                      nombre de messages en DATA
 40 READ A$
 50 FOR J=1 TO LEN(A$)
 60 POKE AD,ASC(MID$(A$,J,1))
 70 AD=AD+1
 80 NEXT J
 90 POKE AD,0 : REM FIN DE MESSAGE
100 AD=AD+1
110 NEXT I
120 DATA "NEXT SANS FOR","ERREUR DE SYNTAXE","RETURN SAN
    GOSUB"
```

7.2.6 Conversion d'une variable en MAJUSCULE.

Cette petite routine permet de transformer le contenu d'une variable chaine en caractères majuscules, elle utilise la routine de la ROM qui transforme les MOTS CLES en majuscules

Programme BASIC.

```
 10 CLEAR 200,&HF000
 20 DEFUSR=&HF000
 30 FOR I=&HF000 to &HF014
 40 READ A$
 50 POKE I,VAL("&H"+A$)
 60 NEXT I
 70 INPUT A$ : REM saisie de la variable à transformer
 80 L=USR(VARPTR(A$)) : REM transformation
 90 PRINT A$
100 GOTO 70 : REM on recommence
110 DATA CD,8A,2F,46,23,5E,23,56,EB,78,FE,00,CB,CD,A9,4E
120 DATA 77,23,05,18,F4
```

Programme en langage machine contenu dans le programme BASIC.

```
F000   CD 8A 2F    DEBUT   CALL    02F8AH   HL=VARPTR de A$
F003   46                  LD      B,(HL)   B=LEN de A$
F004   23                  INC     HL
F005   5E                  LD      E,(HL)
F006   23                  INC     HL
F007   56                  LD      D,(HL)   DE=adresse de A$
F008   EB                  EX      DE,HL    HL=adresse de A$
F009   78          LOOP    LD      A,B      A=LEN de A$
F00A   FE 00               CP      0        Est-ce fini ?
F00C   CB                  RET     Z        Oui retour BASIC
F00D   CD A9 4E            CALL    04EA9H   non conversion en
                                            majuscule du car
                                            pointé par HL.
F010   77                  LD      (HL),A   REMISE dans HL
F011   23                  INC     HL       CARACTERE suivant
F012   05                  DEC     B        LEN=LEN-1
F013   18 F4               JR      LOOP     On recommence
```

7.2.7 Positionnement du CAPS LOCK par programme.

Il suffit de charger le paramètre de la région de communicatio situé à l'adresse 0FCABH avec 0 pour le mode minuscule et avec 255 pour le mode majuscule. En outre il faut éteindre ou allume le témoin à l'aide du BIT 6 du PORT C du PPI.



Utilisons le mode positionnement de BIT. (voir 4.3.2)

Pour allumer la lampe : il faut envoyer 00001100 sur le PORT 0ABH et pour l'éteindre 00001101 sur le même PORT.

```
PASSAGE EN MODE MAJUSCULE : OUT &HAB,12 : POKE &HFCAB,255
PASSAGE EN MODE MINUSCULE : OUT &HAB,13 : POKE &HFCAB,0
```

7.2.8 Manipulations avec le VDP.

a) Extinction et allumage de l'écran.

Il suffit de jouer sur le BIT 6 de R1 (VDP(1)).

La fonction XOR permet d'inverser un bit. BIT 6 = 64

Il suffit d'écrire : VDP(1)=VDP(1) XOR 64 pour inverser l'éta de l'écran .

b) Effacement du contenu des touches de fonction et récupérati de la 24 ième ligne.

Il suffit de faire un CALL en 0CCH.

```
10 DEFUSR=&HCC : L=USR(0)
```

c) Rappel des touches de fonction.

Il suffit de faire un CALL en 0C9H après avoir mis un nombre différent de 0 à l'adresse 0F3DEH.

```
10 DEFUSR=&HC9 : POKE &HF3DE,1 : L=USR(0)
```

d) Curseur vivant (LIVE CURSOR)

Pour avoir une zone écran reprenant la lettre ou le symbole sur lequel se trouve le curseur, il suffit de faire :

PRINT CHR$(255) à l'endroit de votre choix.

e) Pour imprimer certains caractères graphiques qui n'ont pas de code ASCII il suffit de faire :

PRINT CHR$(1);CHR$(n) avec n compris entre 65 et 95.

f) Modification du facteur d'agrandissement d'un SPRITE
sans toucher au mode SCREEN et sans effacer le SPRITE.

Il suffit de modifier le BIT 0 du registre 1 du VDP

```
10 VDP(1)=VDP(1) XOR 1 : REM BASCULE de normal à 2 X et
                             vice-versa.
```

g) Copie de la VIDEORAM dans la mémoire.

Une copie de la VIDEORAM vers la mémoire centrale peut être très utile pour sauvegarder un écran.

La VIDEORAM occupant 16K, une copie complète ne peut être réalisée que sur un système possédant au moins 32K RAM.

Nous réserverons la mémoire de l'adresse 8000H à 87FFH au BASIC et nous copierons la VIDEORAM de l'adresse 8800H à l'adresse C7FFH.
Enfin, la mémoire à partir de F000H est reservée au programme en langage machine.

Programme BASIC :

```
10 CLEAR 100,&H8800
20 FOR I=&HF000 to &HF00B
30 READ A$
40 POKE I,VAL("&H"+A$)
50 NEXT I
60 DEFUSR=&HF000
70 A=USR(0)
80 DATA 21,00,00,11,00,88,01,00,40,C3,59,00
```

Programme en langage machine contenu dans le programme BASIC

```
F000   21 00 00   LD   HL,0000H   HL pointe sur VIDEORAM
F003   11 00 88   LD   DE,8800H   DE pointe sur début RAM
F006   01 00 40   LD   BC,4000H   BC=nombre d'octets
F008   C3 59 00   JP   0059H      VECTEUR ROM
```

Nous avons utilisé un vecteur ROM (59H) qui réalise toute l'opération. A la place d'un CALL nous avons effectué un JP
Le RET de la ROUTINE devient alors le RET de retour au BASIC.



h) Changement de page écran en mode TEXTE.

En mode TEXTE (SCREEN 0), il peut être utile de disposer de plusieurs pages écran. Ce mode n'étant pas gourmand en mémoire, nous disposons de 14 pages.

Rappel : La table qui contient le contenu de l'écran est la TNP située en standard à l'adresse 0.
La TGP contient le jeu de caractères et occupe les adresses de 2048 (800H) à 4095 (0FFFH).

Nous pouvons donc disposer la TNP en 0, en 400H, et dans toutes les adresses supérieures à 0FFFH.

Possibilités :

Rappel : le registre 2 contient l'adresse de la TNP, celle-ci doit se trouver à un multiple de 400H (1024).

| ADRESSE | VAL R2 | ADRESSE | VAL R2 | ADRESSE | VAL R2 |
|---|---|---|---|---|---|
| 0000H | 0 | 1C00H | 7 | 3000H | 12 |
| 0400H | 1 | 2000H | 8 | 3400H | 13 |
| 1000H | 4 | 2400H | 9 | 3800H | 14 |
| 1400H | 5 | 2800H | 10 | 3C00H | 15 |
| 1800H | 6 | 2C00H | 11 | | |

Pour changer de page, il suffit de :

1) Changer la valeur de BASE(0)
2) Changer la valeur du registre 2 ( VDP(2) ).
3) Modifier la valeur 0F923H en y mettant la valeur de l'octet le plus significatif de l'adresse

Exemple : Commuter sur la page située en 2000H.

```
10 BASE(0)=&H2000 : REM point 1
20 VDP(2)=8       : REM 8 est la valeur de R2 pour 2000H
30 POKE &HF923,&H20 : REM Octet le plus significatif.
```

## 7.3 Programme de construction automatique de DATA.

Ce programme qui n'utilise pas de routine en langage machine, permet de construire automatiquement des lignes de DATA.

Le programme demande l'adresse de début et l'adresse de fin de la zone mémoire à sauvegarder.

La ligne 60070 donne le début de votre mémoire, si vous ne possédez que 16K mémoire remplacer 8001 par C001.

La mémoire est 'fouillée' pour retrouver l'adresse des 2 * suivis de 2 /. Ils indiquent le début de la ligne DATA (ligne 60090). Une fois cette adresse déterminée, les données à introduire sont transformées en hexadécimal et 'pokées' à l'intérieur de la ligne DATA. Ensuite la ligne est terminée par ':REM' pour que les * en trop ne gènent pas le programme.

Enfin, le programme constructeur s'efface pour ne laisser que les DATA.

Remarque : Une ligne de DATA doit contenir au moins 225 astérisques (*) et permet de stocker environ 70 valeurs. Vous devez prévoir suffisamment de lignes de DATA (en dupliquant la ligne 50000) pour contenir toutes vos valeurs.

Exemple : Si l'adresse de début de la mémoire à sauver vaut 1000 et l'adresse de fin 1500, il sera nécessaire de dupliquer la ligne 50000 de 10 en 10 jusqu'à la ligne 50070 (8X70=560 valeurs).



Programme :

```
50000 DATA **//************************************************
***************************************************************
***************************************************************
***************************************************************

60000 REM DATAPACK
60010 CLEAR 1000,&H9000
60020 DEFINT A-Z
60030 INPUT"ADRESSE DE DEPART ";S
60040 INPUT"ADRESSE DE FIN ";E
60050 CLS
60060 PRINT"JE TRAVAILLE"
60070 A=&H8001
60080 FOR I=A TO A+3000
60090 IF PEEK(I)<>42 THEN NEXT ELSE IF PEEK(I+1)<>42 THEN NEXT  EL
IF PEEK(I+2)<>47 THEN NEXT ELSE IF PEEK(I+3)<>47 THEN NEXT ELSE B=
60100 FOR I=S TO E
60110 C=PEEK(I)
60120 C$=HEX$(C)
60130 IF LEN(C$)=1 THEN C$="0"+C$
60135 PRINT C$;" ";
60140 C1$=LEFT$(C$,1)
60150 C2$=RIGHT$(C$,1)
60160 C1=ASC(C1$)
60170 C2=ASC(C2$)
60180 POKE B,C1:POKE B+1,C2
60190 FL=FL+3:B=B+3:IF FL>220 THEN GOSUB 60250
60200 IF FL>0 THEN POKE B-1,44
60210 NEXT I
60220 GOSUB 60250
60230 GOTO 60300
60240 *
60250 POKE B-1,32:POKE B,58:POKE B+1,143
60260 FOR J=B TO B+220
60270 IF PEEK(J)<>42 THEN NEXT ELSE IF PEEK(J+1)<>42  THEN NEXT EL
IF PEEK(J+2)<>47 THEN NEXT ELSE IF PEEK(J+3)<>47 THEN NEXT ELSE B=
60280 FL=0
60290 RETURN
60300 PRINT:PRINT:PRINT"TERMINE"
60320 DELETE 60000-60320
```

## 7.4 Passage d'arguments multiples à une fonction USR.

Il est parfois nécessaire de passer plusieurs arguments à une fonction USR.

Exemple: la réalisation d'une routine de 'BLOCK MOVE' (recopie d'une zone mémoire d'une adresse à une autre) demande le passage de 3 arguments: l'adresse de départ, l'adresse d'arrivée et le nombre d'octets à copier.

Avec le programme suivant, il suffira d'écrire :

L=USR(X) OR USR(Y) OR USR(Z)

Où X est l'adresse de départ, Y l'adresse d'arrivée et Z le nombre d'octets à copier.

Cette routine est écrite pour l'USR0. Pour un autre USR, il faut adapter l'adresse de stockage de l'USR dans la région de communication

Programme en langage machine :

Ce programme contient la routine de passage d'arguments multiples ainsi qu'un programme de démonstration de son utilisation (BLOCK MOVE). Le programme de démonstration commence au label START.

```
D000  CD 8A 2F    DEBUT   CALL   02F8AH       ARGUMENT DANS HL
D003  DD 2A 9A F3         LD     IX,0F39AH    IX=ADRESSE DEFUSR0
D007  DD 75 30            LD     (IX+30H),L   SAUVE ARGUMENT DANS
D00A  DD 74 31            LD     (IX+31H),H   ZONE DE STOCKAGE
D00D  DD 34 09            INC    (IX+9)       ADDITIONNE 2 AU
D010  DD 34 09            INC    (IX+9)       PREMIER POINTEUR
D013  DD 34 0C            INC    (IX+12)      ADDITIONNE 2 AU
D016  DD 34 0C            INC    (IX+12)      SECOND POINTEUR
D019  DD 7E 09            LD     A,(IX+9)
D01C  06 30               LD     B,30H
D01E  90                  SUB    B            A=ARGUMENT *2
D01F  DD 46 2F            LD     B,(IX+2FH)   B=ARGUM2 * 2
D022  90                  SUB    B
D023  28 06               JR     Z,SUITE      C'EST FINI
D025  C9                  RET                 RETOUR AU BASIC
```



```
D026  DD 36 09 30 SUITE   LD     (IX+9),30H
D02A  DD 36 0C 31         LD     (IX+12),31H
D02E  18 06               JR     START        EXECUTION
D030  00 00               DEFW   0            ZONE ARG 1
D032  00 00               DEFW   0            ZONE ARG 2
D034  00 00               DEFW   0            ZONE ARG 3
D036  DD 6E 30    START   LD     L,(IX+30H)   HL = ARG1
D039  DD 66 31            LD     H,(IX+31H)
D03C  DD 5E 32            LD     E,(IX+32H)   DE = ARG2
D03F  DD 56 33            LD     D,(IX+33H)
D042  DD 4E 34            LD     C,(IX+34H)   BC = ARG3
D045  DD 46 35            LD     B,(IX+35H)
D048  ED B0               LDIR                MOVE
D04A  C9                  RET                 RETOUR AU BASIC
```

Il est possible de passer plus de 3 arguments en ajoutant lignes de DEFW supplémentaires. Il suffit de déplacer le START JP START en fonction du nombre d'arguments.

Ce système est très efficace pour effectuer des ap multi-arguments.

Il présente les avantages et les inconvénients suivants:

1) Vous devez connaître le numéro de l'USR.
2) La routine occupe 42 octets + 2 octets par argument.
3) La routine se modifie dynamiquement pendant son exécutio nombre d'arguments doit être respecté.
4) La routine est indépendante de la position mémoire.
5) La routine accepte 25 arguments sans problème.

Remarque : Le programme BASIC correspondant n'est pas donn est très facile à établir et arrivé a ce stade du livre vous être capable de le faire seul.

## 7.5 Démonstration de la technique de la variable tableau.

Nous allons réaliser le programme de 'BLOCK MOVE' de l'exemple précédent au moyen d'une variable tableau.

Le passage d'arguments est très facile avec les variables tableau. La seule condition à respecter est la position de l'argument dans le programme, il doit correspondre à un multiple de 2 octets en partant du début du programme. Il est donc parfois nécessaire d'introduire des instructions NOP.

Rappel : programme de 'BLOCK MOVE'.

```
XX00  21 VV VV       LD      HL,ARG1
XX03  11 WW WW       LD      DE,ARG2
XX06  01 ZZ ZZ       LD      BC,ARG3
XX09  ED B0          LDIR
XX0B  C9             RET
```

Les arguments ARG1 et ARG3 ne se trouvent pas à un multiple de 2 octets par rapport au début du programme. Le programme doit être transformé en :

```
XX00  00             NOP
XX01  21 VV VV       LD      HL,ARG1
XX04  00             NOP
XX05  11 WW WW       LD      DE,ARG2
XX08  00             NOP
XX09  01 ZZ ZZ       LD      BC,ARG3
XX0C  ED B0          LDIR
XX0E  C9             RET
```

Les octets en remplacant VV,WW et ZZ par 00 deviennent :

00 21 00 00 00 11 00 00 00 01 00 00 ED B0 C9

On prend les octets 2 par 2 et on calcule la valeur des 2 octets en binaire signé par l'instruction PRINT HEX$(XXYY) où XX est le second octet et YY le premier



Résultat :

```
00 21 -> HEX$(2100) = 8448
00 00 ->            = 0
00 11 -> HEX$(1100) = 4352
00 00 ->            = 0
00 01 -> HEX$(0100) =  256
00 00 ->            = 0
ED B0 -> HEX$(B0ED) =45293 -> 45293-65536 (binaire signé) = -20243
C9 00 -> HEX$(00C9) =  201
```

Il suffit de charger ces valeurs dans une variable table entière, élément par élément, dans l'ordre et en commençant p l'élément 0. Ensuite, il suffit de remplacer les valeurs 0 (VV,WW ZZ) par les valeurs saisies en cours de programme.

Programme:

```
10 DEFINT A-Z : J=0 : A$=""
20 VT(0)=8448 : VT(1)=0 : VT(2)=4352 : VT(3)=0 : VT(4)=256
30 VT(5)=0 : VT(6) = -20243 : VT(7)=201
40 CLS
50 INPUT"ADRESSE DE DEPART ";VT(1)
60 INPUT"ADRESSE D'ARRIVEE ";VT(3)
70 INPUT"NOMBRE D'OCTETS   ";VT(5)
80 DEFUSR=VARPTR(VT(0))
90 J=USR(0)
```

## 7.6 Programme de saisie de fonctions.

Ce programme est assez spécial, il peut être très utile à tous ceux qui utilisent l'ordinateur dans un but scientifique.

Le BASIC a une grande lacune: on ne peut pas entrer une fonction en mode conversationnel (INPUT) et effectuer des calculs sur cette fonction.

Le présent programme remédie à cet état de fait.

Le programme contient une partie de démonstration que vous pouvez adapter à vos besoins (lignes 200 à 230).

Fonctionnement :

Au départ, la variable Z$ contient un certain nombre d'astérisques (*). La variable X$ contient la fonction à calculer. On détermine le VARPTR des 2 variables et on appelle un programme en langage machine par une instruction USR en passant comme argument le VARPTR de Z$. Le programme en langage machine retourne directement au BASIC après le premier appel, c'est le second appel du même programme (avec comme argument le VARPTR de X$) qui déclenche le processus.

L'appel double est réalisé par la ligne 140. Le programme en langage machine transforme la ligne de programme où se trouve Z$. Il en fait une fonction définie par l'utilisateur de la forme DEF FNY(X)= suivi de la fonction à calculer.

Le programme en langage machine se charge de transformer la fonction en code compréhensible par l'interpréteur BASIC.

Après l'appel, il est nécessaire de repasser sur la nouvelle ligne créée pour que la fonction soit connue du BASIC, c'est le but du sémaphore de passage F qui aiguille le second passage.

Enfin après la phase de calcul, il est nécessaire de remettre la variable Z$ dans son état initial, c'est le but de l'appel USR avec l'argument 0 de la ligne 240. C'est la valeur de l'argument qui déclenche le processus.



Programme Basic :

```
10  REM  X ET Y SONT RESERVES POUR LA FONCTION Y=F(X)
20  CLEAR 200,&HD000
30  FOR I=&HD000 TO &HD088
40  READ A$
50  POKE I,VAL("&H"+A$)
60  NEXT I
70  CLS
80  F=0 ; REM SEMAPHORE DE PASSAGE
90  Z$="******************************************"
95  IF F=1 THEN GOTO 200 ; REM SECOND PASSAGE
100 LINEINPUT"ENTREZ LA FONCTION A ANALYSER SOUS LA FORME F(X) ";X$
110 L=VARPTR(Z$)
120 K=VARPTR(X$)
130 DEFUSR=&HD000
140 M = USR(L) OR USR(K) ; REM RESPECTEZ L'ORDRE D'APPEL
150 F=1
160 GOTO 90
200 PRINT
210 INPUT"ENTREZ UNE VALEUR VALEUR DE X ";R
220 PRINT
230 PRINT"AU POINT ";R;" LA FONCTION VAUT ";FNY(R)
240 M=USR(0) ; REM REMISE DE Z$ A L'ETAT INITIAL
250 GOTO 80 ; REM ON RECOMMENCE
300 DATA CD,8A,2F,AF,BC,20,07,2A,8D,D0,CD,75,D0,C9,3A,8C
310 DATA D0,FE,01,28,09,22,8D,D0,3E,01,32,8C,D0,C9,EB,2A
320 DATA 8D,D0,23,4E,23,46,0B,0B,0B,0B,60,69,36,97,23,36
330 DATA DE,23,36,59,23,36,28,23,36,58,23,36,29,23,36,EF
340 DATA 23,1A,32,8C,D0,13,1A,4F,13,1A,47,E5,C5,E1,11,90
350 DATA D0,3A,8C,D0,4F,06,00,ED,B0,AF,12,21,90,D0,CD,B2
360 DATA 42,E1,11,1F,F4,1A,FE,00,28,05,77,23,13,18,F6,36
370 DATA 3A,23,36,8F,C9,2B,2B,5E,23,23,23,4E,23,46,60,69
380 DATA 2B,36,22,2B,36,EF,2B,36,24,2B,73,C9
```

Programme en Assembleur contenu dans le programme Basic.

```
                    GETARG  EQU     02F8AH       HL=VALEUR DE USR
D000  CD 8A 2F      DEBUT   CALL    GETARG       PRISE DE L'ARGUMENT
                                                 DANS HL
D003  AF                    XOR     A            A<>0
D004  BC                    CP      H            H=0?
D005  20 07                 JR      NZ,NREST     NON, C'EST UN VARPTR
D007  2A 8D D0              LD      HL,(STK+1)   OUI, REPASSE ADRESSE
                                                 DE L (VARPTR Z$)
D00A  CD 75 D0              CALL    RESTOR       RESTAURATION LIGNE
```

```
D00D  C9                  RET                     RETOUR AU BASIC
D00E  3A 8C D0  NREST     LD      A,(STK)         EST-CE LE PREMIER
D011  FE 01               CP      1               PASSAGE ?
D013  28 09               JR      Z,SUITE         NON ALORS SUITE
D015  22 8D D0            LD      (STK+1),HL      OUI SAUVEGARDE DU
D018  3E 01               LD      A,1             VARPTR ET MISE A 1 DU
D01A  32 8C D0            LD      (STK),A         FLAG DE PASSAGE
D01D  C9                  RET                     RETOUR A BASIC
D01E  EB        SUITE     EX      DE,HL           SAUVE VARPTR X$->DE
D01F  2A 8D D0            LD      HL,(STK+1)      CHARGE VARPTR Z$
D022  23                  INC     HL              CHARGE BC AVEC
D023  4E                  LD      C,(HL)          L'ADRESSE DE Z$
D024  23                  INC     HL
D025  46                  LD      B,(HL)
D026  0B                  DEC     BC              RECULE BC DE 4 POUR
D027  0B                  DEC     BC              POINTER SUR LA LETTRE
D028  0B                  DEC     BC               Z (Z$=")
D029  0B                  DEC     BC
D02A  60                  LD      H,B             HL = NOUVELLE ADRESSE
D02B  69                  LD      L,C              (LETTRE Z)
D02C  36 97               LD      (HL),151        CODE DE DEF
D02E  23                  INC     HL              POSITION SUIVANTE
D02F  36 DE               LD      (HL),222        CODE DE FN
D031  23                  INC     HL              POSITION SUIVANTE
D032  36 59               LD      (HL),89         CARACTERE Y
D034  23                  INC     HL
D035  36 28               LD      (HL),28H        CARACTERE (
D037  23                  INC     HL
D038  36 58               LD      (HL),88         CARACTERE X
D03A  23                  INC     HL
D03B  36 29               LD      (HL),29H        CARACTERE )
D03D  23                  INC     HL
D03E  36 EF               LD      (HL),0EFH       CODE DE =
D040  23                  INC     HL
D041  1A                  LD      A,(DE)          A = LONGUEUR DE X$
D042  32 8C D0            LD      (STK),A         SAUVE LONGUEUR DE X$
D045  13                  INC     DE              CHARGE BC AVEC
D046  1A                  LD      A,(DE)          L'ADRESSE DE X$
D047  4F                  LD      C,A
D048  13                  INC     DE
D049  1A                  LD      A,(DE)
D04A  47                  LD      B,A
D04B  E5                  PUSH    HL              SAUVE CONTENU DE HL
D04C  C5                  PUSH    BC              SAUVE CONTENU DE BC
D04D  E1                  POP     HL              HL = CONTENU DE BC
D04E  11 90 D0            LD      DE,AREA         DE POINTE SUR TAMPON
D051  3A 8C D0            LD      A,(STK)         A= LONGUEUR DE X$
D054  4F                  LD      C,A             BC = LONGEUR DE X$
D055  06 00               LD      B,0
D057  ED B0               LDIR                    COPIE DE X$ -> TAMPON
D059  AF                  XOR     A               A=0
D05A  12                  LD      (DE),A          TERMINE TAMPON PAR 00
```



```
D05B  21 90 D0            LD      HL,AREA         HL=DEBUT TAMPON
D05E  CD B2 42            CALL    04EA9H          ROUTINE PRINCIPALE OU
                                                  CONVERTIT LA  CHAI
                                                  POINTEE PAR HL EN CO
                                                  LE  RESULTAT  EST
                                                  L'ADRESSE
D061  E1                  POP     HL              RECUPERE  HL  (ADRES
                                                  APRES  LE  SIGNE  =
                                                  DEFFNY(X)=)
D062  11 1F F4            LD      DE,0F41FH       DE   =   ADRESSE
                                                  RESULTAT DE LA ROUTI
D065  1A        NEXT      LD      A,(DE)          EST-CE LA FIN DU
D066  FE 00               CP      00              RESULTAT
D068  28 05               JR      Z,FIN           OUI SAUT A FIN
D06A  77                  LD      (HL),A          NON    TRANSFERT
                                                  CARACTERE
D06B  23                  INC     HL              CARACTERE SUIVANT
D06C  13                  IND     DE
D06D  18 F6               JR      NEXT
D06F  36 3A     FIN       LD      (HL),3AH        CARACTERE :
D071  23                  INC     HL              POSITION SUIVANTE
D072  36 8F               LD      (HL),143        CODE DE REM
D074  C9                  RET                     RETOUR AU BASIC
D075  2B        RESTOR    DEC     HL              REPRISE DU NOM DE
D076  2B                  DEC     HL              LA VARIABLE (Z)
D077  5E                  LD      E,(HL)          SAUVE DANS E
D078  23                  INC     HL              POSITIONNE HL SUR LE
D079  23                  INC     HL              BON ENDROIT
D07A  23                  INC     HL
D07B  4E                  LD      C,(HL)          BC = ADRESSE DU
D07C  23                  INC     HL              CONTENU DE LA
D07D  46                  LD      B,(HL)          VARIABLE (Z$)
D07E  60                  LD      H,B             HL=BC
D07F  69                  LD      L,C
D080  2B                  DEC     HL              RECULE HL
D081  36 22               LD      (HL),22H        ECRIT LE "
D083  2B                  DEC     HL
D084  36 EF               LD      (HL),0EFH       ECRIT LE SIGNE =
D086  2B                  DEC     HL
D087  36 24               LD      (HL),24H        ECRIT LE SIGNE $
D089  2B                  DEC     HL
D08A  73                  LD      (HL),E          ECRIT  LE  NOM  DE
                                                  VARIABLE
D08B  C9                  RET                     RETOUR AU BASIC
D08C            STK       EQU     $
D090            AREA      EQU     $+4
                          END
```

## 7.7 Addition d'un vecteur.

Dans le but de démontrer l'utilisation des appels des routines mathématiques contenues dans la ROM BASIC, voici un programme qui réalise la somme de tous les éléments d'une variable tableau. Vous pourrez en apprécier la vitesse d'exécution, la somme de 1000 éléments s'effectuant en un peu plus d'une seconde.

Programme BASIC:

```
10 CLEAR 200,&HD000
20 REM
30 REM CHARGEMENT DU PROGRAMME EN LANGAGE MACHINE
40 REM
50 FOR I=&HD000 TO &HD03A
60 READ A$
70 POKE I,VAL("&H"+A$)
80 NEXT I
90 REM
100 REM INITIALISATION DES ELEMENTS
120 SM=0 : REM CETTE VARIABLE RECEVRA LE RESULTAT DU CALCUL
130 DIM A(999)
140 DEFUSR=&HD000
150 REM
160 REM CREATION DES 1000 VALEURS
180 PRINT"PATIENCE, JE CREE MES 1000 VALEURS"
190 FOR I=0 TO 999
200 A(I)=RND(I)
210 PRINT I;
220 NEXT I
230 REM
240 REM PREPARATION DU VARPTR DE LA VARIABLE DE RECEPTION
260 V=VARPTR(SM)
270 V$=HEX$(V)
280 POKE &hD030,VAL("&H"+RIGHT$(V$,2))
290 POKE &HD031,VAL("&H"+LEFT$(V$,2))
300 J=0 : REM INITIALISATION DE J POUR NE PAS PERTURBER LA TVT
310 REM
320 REM ADDITION ET CALCUL DU TEMPS
340 CLS
350 PRINT"DEBUT DE L'ADDITION"
360 TIME=0
370 J=USR(VARPTR(A(0)))
```



```
380 T2=TIME
390 PRINT"RESULTAT : ";SM
400 PRINT
410 PRINT"TEMPS : ";T2/50
420 REM
430 REM DATA DU PROGRAMME EN LANGAGE MACHINE
450 DATA CD,8A,2F,E5,2B,46,2B,4E,D1,D5,C5,3E,08,32,63,F6
460 DATA 21,F6,F7,CD,F3,2E,C1,D1,0B,79,B0,28,12,21,08,00
470 DATA 19,E5,C5,EB,21,47,F8,CD,F3,2E,CD,9A,26,18,E7,11
480 DATA 00,00,21,F6,F7,01,08,00,ED,B0,C9
```

Programme assembleur contenu dans le programme BASIC:

```
D000  CD 8A 2F   DEBUT  CALL  02F8AH      HL = VARPTR A(0)
D003  E5                PUSH  HL          SAUVE HL
D004  2B                DEC   HL          BC=NOMBRE D'ELEMENTS
D005  46                LD    B,(HL)
D006  2B                DEC   HL
D007  4E                LD    C,(HL)
D008  D1                POP   DE          DE = VARPTR A(0)
D009  D5                PUSH  DE          SAUVE DE
D00A  C5                PUSH  BC          SAUVE BC
D00B  3E 08             LD    A,8         TYPE DOUBLE PRECISION
D00D  32 63 F6          LD    (0F663H),A  POSITIONNE STD A 8
D010  21 F6 F7          LD    HL,0F7F6H   HL = ADRESSE ACCUM1
D013  CD F3 2E          CALL  02EF3H      COPIE (DE) -> ACCUM1
D016  C1         LOOP   POP   BC          BC=NOMBRE D'ELEMENTS
D017  D1                POP   DE          DE =VARPTR A(n)
D018  0B                DEC   BC          BC=BC-1
D019  79                LD    A,C         A=NOMBRE RESTANT
D01A  B0                OR    B           EST-CE 0 ?
D01B  28 12             JR    Z,FIN       OUI SAUT A FIN
D01D  21 08 00          LD    HL,8        POSITIONNE HL SUR
D020  19                ADD   HL,DE       L'ELEMENT SUIVANT
D021  E5                PUSH  HL          SAUVE HL
D022  C5                PUSH  BC          SAUVE BC
D023  EB                EX    DE,HL       DE=ELEMENT SUIVANT
D024  21 47 F8          LD    HL,0F847H   HL = ADRESSE ACCUM2
D027  CD F3 2E          CALL  02EF3H      COPIE (DE) -> ACCUM2
D02A  CD 9A 26          CALL  0269AH      ACCUM1=ACCUM1+ACCUM2
D02D  18 E7             JR    LOOP        SUIVANT
D02F  11 00 00   FIN    LD    DE,0000       =VARPTR VARIABLE D
                                          RECEPTION (POKE)
D032  21 F6 F7          LD    HL,0F7F6H   HL = ADRESSE ACCUM1
D035  01 08 00          LD    BC,8        BC = 8 OCTETS
D038  ED B0             LDIR              TRANSFERT (HL) -> (DE
D03A  C9                RET               RETOUR AU BASIC
```

## 7.8 Conversion du clavier en AZERTY.

Pour convertir le clavier en AZERTY, il faut intercepter le vecteur de lecture du clavier à l'adresse OFDCCH. Quand ce vecteur vous donne la main, le registre C contient le numéro de la ligne (numérotée de 0 à 8) de la touche enfoncée multiplié par 8 + le numéro de la colonne (numérotée de 0 à 7) de cette touche. Pour toute information complémentaire référez-vous à la structure du clavier au point 4.4.

Les touches à inverser sont : le A et le Q , le Z et le W, le M et le ; .

A est à l'intersection de la ligne 2 et de la colonne 6
A = 2 * 8 + 6 = 22 = 16H

Q est à l'intersection de la ligne 4 et de la colonne 6
Q = 4 * 8 + 6 = 38 = 26H

Z est à l'intersection de la ligne 5 et de la colonne 7
Z = 5 * 8 + 7 = 47 = 2FH

W est à l'intersection de la ligne 5 et de la colonne 4
W = 5 * 8 + 4 = 44 = 2CH

M est à l'intersection de la ligne 4 et de la colonne 2
M = 4 * 8 + 2 = 34 = 22H

; est à l'intersection de la ligne 1 et de la colonne 7
; = 1 * 8 + 7 = 15 = 0FH

REMARQUE : L'installation du JP (C3H) doit être la dernière instruction du programme, l'interception du vecteur ne pouvant se faire que lorsque tout est en place.



Programme BASIC:

```
10 REM ce programme modifie le clavier en AZERTY
20 CLEAR 200,&HF000
30 AD=&HFDCC          ; REM vecteur du clavier
40 POKE AD+1,0        ; REM partie basse de F000H
50 POKE AD+2,&HF0     ; REM partie haute de F000H
60 FOR I=&HF000 TO &HF02B
70 READ A$
80 POKE I,VAL("&H"+A$)
90 NEXT I
100 DATA 79,FE,16,20,03,0E,26,C9,FE,26,20,03,0E,16,C9,FE
110 DATA 2C,20,03,0E,2F,C9,FE,2F,20,03,0E,2C,C9,FE,22,20
120 DATA 03,0E,0F,C9,FE,0F,20,03,3E,22,4F,C9
130 POKE AD,&HC3      ; REM installation du JP.
```

Programme en assembleur contenu dans le programme BASIC.

```
F000  79      DEBUT  LD   A,C       C CONTIENT LA TOUCHE
F001  FE 16          CP   016H      EST-CE A ?
F003  20 03          JR   NZ,PASA   NON PAS A
F005  0E 26          LD   C,026H    OUI REMPLACE PAR Q
F007  C9             RET            RETOUR
F008  FE 26   PASA   CP   026H      EST-CE Q ?
F00A  20 03          JR   NZ,PASQ   NON PAS Q
F00C  0E 16          LD   C,016H    OUI REMPLACE PAR A
F00E  C9             RET
F00F  FE 2C   PASQ   CP   02CH      EST-CE W ?
F011  20 03          JR   NZ,PASW   NON PAS W
F013  0E 2F          LD   C,02FH    OUI REMPLACE PAR Z
F015  C9             RET
F016  FE 2F   PASW   CP   02FH      EST-CE Z ?
F018  20 03          JR   NZ,PASZ   NON PAS Z
F01A  0E 2C          LD   C,02CH    OUI REMPLACE PAR W
F01C  C9             RET
F01D  FE 22   PASZ   CP   022H      EST-CE M ?
F01F  20 03          JR   NZ,PASM   NON PAS M
F021  0E 0F          LD   C,0FH     OUI REMPLACE PAR ;
F023  C9             RET
F024  FE 0F   PASM   CP   0FH       EST-CE ; ?
F026  20 03          JR   NZ,PASP   NON PAS ;
F028  3E 22          LD   A,022H    OUI REMPLACE PAR M
F02A  4F      PASP   LD   C,A       RESTAURE C
F02B  C9             RET            FIN
```

## 7.9 Commutation des SLOTS.

Dans cette section, nous allons nous livrer à quelques manipulations avec les slots.

REMARQUE : Il faut au moins 32 K RAM pour la première manipulation et 64 K RAM pour les suivantes.

1ère manipulation : copie de la mémoire d'un slot en mémoire centrale.

Nous supposons posséder une cartouche qui se charge dans le slot 3 du bank 0 (adresse 0000 à 3FFF) et nous désirons en copier le contenu en mémoire RAM de l'adresse 9000H à l'adresse CFFFH.

Programme BASIC :

```
10 CLEAR 200,&H9000
20 FOR I=&HF000 TO &HF014
30 READ A$:POKE I,VAL("&H"+A$)
40 NEXT I
50 DEFUSR0=&HF000
60 L=USR(0)
70 DATA F3,3E,03,D3,A8,21,00,00,11,00,90,01,00,40,ED,B0
80 DATA 3E,00,D3,A8,C9
```

Programme en assembleur :

```
F000  F3        DI                   PAS D'INTERRUPTION
F001  3E 03     LD    A,3            SLOT 3 DU BANK 0
F003  D3 A8     OUT   (0A8H),A       SELECTION DU SLOT
F005  21 00 00  LD    HL,0000H       HL = DEPART
F008  11 00 90  LD    DE,09000H      DE = ARRIVEE
F00B  01 00 40  LD    BC,04000H      BC = COMPTEUR
F00E  ED B0     LDIR                 TRANSFERT
F010  3E 00     LD    A,0            SLOT 0 = ROM BASIC
F012  D3 A8     OUT   (0A8H),A       SELECTION
F014  C9        RET                  RETOUR AU BASIC
```

2ème manipulation : Copie de la ROM en RAM. Nécessite 64 K RAM.

La ROM occupe le SLOT 0 du BANK 0 et du BANK 1.
La RAM 32 K occupe les SLOTS 1 du BANK 0 et du BANK 1.
Il suffit donc de copier (par tranche de 16 K) la ROM en haut de mémoire (9000) puis de commuter le SLOT 1 et de copier la mémoire haute dans la mémoire basse :



Utilité : permet de modifier le BASIC. Par exemple, mettre les messages d'erreurs en français (7.2.5).

Utilisation : exécuter le programme BASIC et votre système se trouvera immédiatement en mode 64K RAM, le BASIC MICROSOFT étant chargé de l'adresse 0000H à l'adresse 7FFFH.
Vous pouvez dès lors le modifier au moyen de l'instruction POKE.

Si votre RAM est dans le SLOT 2 (SONY HB75), remplacer 3E05 par 3EAA et 3E00 par 3EA0.

Programme BASIC.

```
10 CLEAR 200,&H9000
20 FOR I=&HF000 TO &HF039
30 READ a$ : POKE I,VAL("&H"+A$)
40 NEXT I
50 DEFUSR=&HF000
60 L=USR(0)
70 DATA F3,21,00,00,11,00,90,01,00,40,ED,B0,3E,05,D3,A8
80 DATA 21,00,90,11,00,00,01,00,40,ED,B0,3E,00,D3,A8,21
90 DATA 00,40,11,00,90,01,00,40,ED,B0,3E,05,D3,A8,21,00
95 DATA 90,11,00,40,01,00,40,ED,B0,C9
```

Programme en langage machine contenu dans le programme BASIC.

```
F000  F3        DEBUT  DI                   PAS D'INTERRUPTION
F001  21 00 00         LD    HL,0000H
F004  11 00 90         LD    DE,9000H
F007  01 00 40         LD    BC,4000H
F00A  ED B0            LDIR                 TRANSFERT 16K BAS
F00C  3E 05            LD    A,5            SELECT SLOT 1 BNK 0
F00E  D3 A8            OUT   (0A8H),A       ET BNK 1.
F010  21 00 90         LD    HL,9000H
F013  11 00 00         LD    DE,0000H
F016  01 00 40         LD    BC,4000H
F019  ED B0            LDIR                 TRANSFERT VERS RAM 0
F01B  3E 00            LD    A,0            SELECT ROM BASIC
F01D  D3 A8            OUT   (0A8H),A
F01F  21 00 40         LD    HL,4000H
F022  11 00 90         LD    DE,9000H
F025  01 00 40         LD    BC,4000H
F028  ED B0            LDIR                 TRANSFERT 16K HAUT
F02A  3E 05            LD    A,5            SELECT SLOT 1 BK 0&1
F02C  D3 A8            OUT   (0A8H),A
F02E  21 00 90         LD    HL,9000H
F031  11 00 40         LD    DE,4000H
F034  01 00 40         LD    BC,4000H
F037  ED B0            LDIR                 TRANSFERT VERS RAM 4000H
F039  C9               RET                  RETOUR AU BASIC EN RAM
```

Comme vous pouvez le remarquer, le transfert se fait en deux étapes. La première fois, on transfère les 16K du bas (0000H-3FFFH) vers la RAM (9000H-CFFFH). Ensuite on commute la RAM à la place de la ROM et on effectue le transfert inverse. La seconde fois, on transfère les 16K du haut (4000H-7FFFH) vers la RAM (9000H-CFFFH). Ensuite on recommute la RAMet on effectue le transfert inverse.

Il est indispensable de procéder en deux passes car les 32K de la RAM ne suffisent pas pour mémoriser les 32K de la ROM en plus de la zone de communication et du programme de transfert.

3ème manipulation : Copie de la VIDEORAM vers un SLOT.

Objet : faire une copie complète de la VIDEORAM (16K) vers un BANK inférieur contenant de la RAM et recopier la RAM vers la VIDEORAM.

Utilité : Faire une sauvegarde rapide et totale de l'écran et ce en n'importe quel mode, et ce sans consommer de mémoire utile au BASIC.

Le programme utilise une fonction USR dont l'argument décide du sens du transfert. Si l'argument est 0, le transfert se fait de la VIDEORAM vers la RAM, sinon, le transfert se fait dans l'autre sens.

Programme BASIC.

```
10 CLEAR 200,&HF000
20 FOR I=&HF000 TO &HF022
30 READ A$ : POKE I,VAL("&H"+A$)
40 NEXT I
50 DEFUSR=&HF000
60 L=USR(0) : REM COPIE DE L'ECRAN DANS LA RAM
70 CLS       : REM EFFACEMENT
80 FOR I=1 TO 2000 : NEXT I : REM ATTENTE
90 L=USR(1) : REM ON RAPPELLE L'ECRAN
95 DATA F3,CD,1F,52,F5,3E,04,D3,A8,21,00,00,11,00,40,01
96 DATA 00,40,F1,A7,20,05,CD,59,00,18,03,CD,45,07,3E,00
97 DATA D3,A8,C9
```



Programme assembleur contenu dans le programme BASIC.

```
F000  F3              DEBUT  DI                   Pas d'interruption
F001  CD 1F 52               CALL  0521FH         Argument dans A
F004  F5                     PUSH  AF             sauve argument
F005  3E 04                  LD    A,4            SELECT SLOT 1 (4000H)
F007  D3 A8                  OUT   (0A8H),A
F009  21 00 00               LD    HL,0000H       HL pointe sur VRAM
F00C  11 00 40               LD    DE,4000H       DE pointe sur RAM
F00F  01 00 40               LD    BC,4000H       BC = compteur
F012  F1                     POP   AF             A=ARGUMENT
F013  A7                     AND   A              Positionne FLAGS
F014  20 05                  JR    NZ,RAM         arg≠0 = écrit VRAM
F016  CD 59 00               CALL  0059H          Lecture VRAM
F019  18 03                  JR    FIN
F01B  CD 45 07        RAM    CALL  0745H          Ecrit VRAM
F01E  3E 00           FIN    LD    A,0            SELECT ROM
F020  D3 A8                  OUT   (0A8H),A
F022  C9                     RET                  RETOUR AU BASIC
```

Ce programme termine les manipulations avec les SLOTS, d'autres possibilités existent. Je vous laisse le soin de les découvrir.

Encore un mot, j'ai utilisé un CALL à l'adresse 0745H en F01BH. Pourquoi ?

Il existe un vecteur en 005CH qui branche à l'adresse 0744H Malheureusement la première instruction à cette adresse inverse HL et DE, pour ne pas devoir modifier les contenus de HL et DE (0F009H-0F00EH) il était plus simple de ne pas utiliser le vecteur et de se brancher une adresse plus loin c-à-d en 0745H.

## 7.10 Supertélécran.

Voivi un programme sans prêtention qui vous permettra d'analyser le fonctionnement des principales fonctions graphiques.

Ce programme permet de dessiner des points, des lignes ou des rectangles et de peindre des surfaces dans une couleur au choix.

L'instrument de dessin peut être soit le clavier, soit le joystick suivant votre choix.

Pour tracer un point, il suffit de pousser sur la barre d'espace ou d'appuyer le bouton de tir.

Pour changer de mode, il suffit de déplacer le réticule vers la droite de l'écran en face du nouveau mode choisi et de pousser la barre d'espace ou le bouton de tir.

Le rectangle noir inférieur indique le mode choisi.

Le rectangle noir supérieur ne sert que dans les modes LINE et BOX. Il indique si on marque l'origine (ORG) ou la destination (DES) de la figure.

A l'aide du programme de la page suivante, le lecteur pourra exercer ses talents de dessinateur informatisé. Les traditionnels tableau et craies des enfants se trouvent ici avantageusement remplacés. En effet, les enfants seront très fiers de créer un dessin à l'aide de l'ordinateur et la lessive se verra quelque peu allégée, le clavier étant nettement moins salissant que les craies!!!

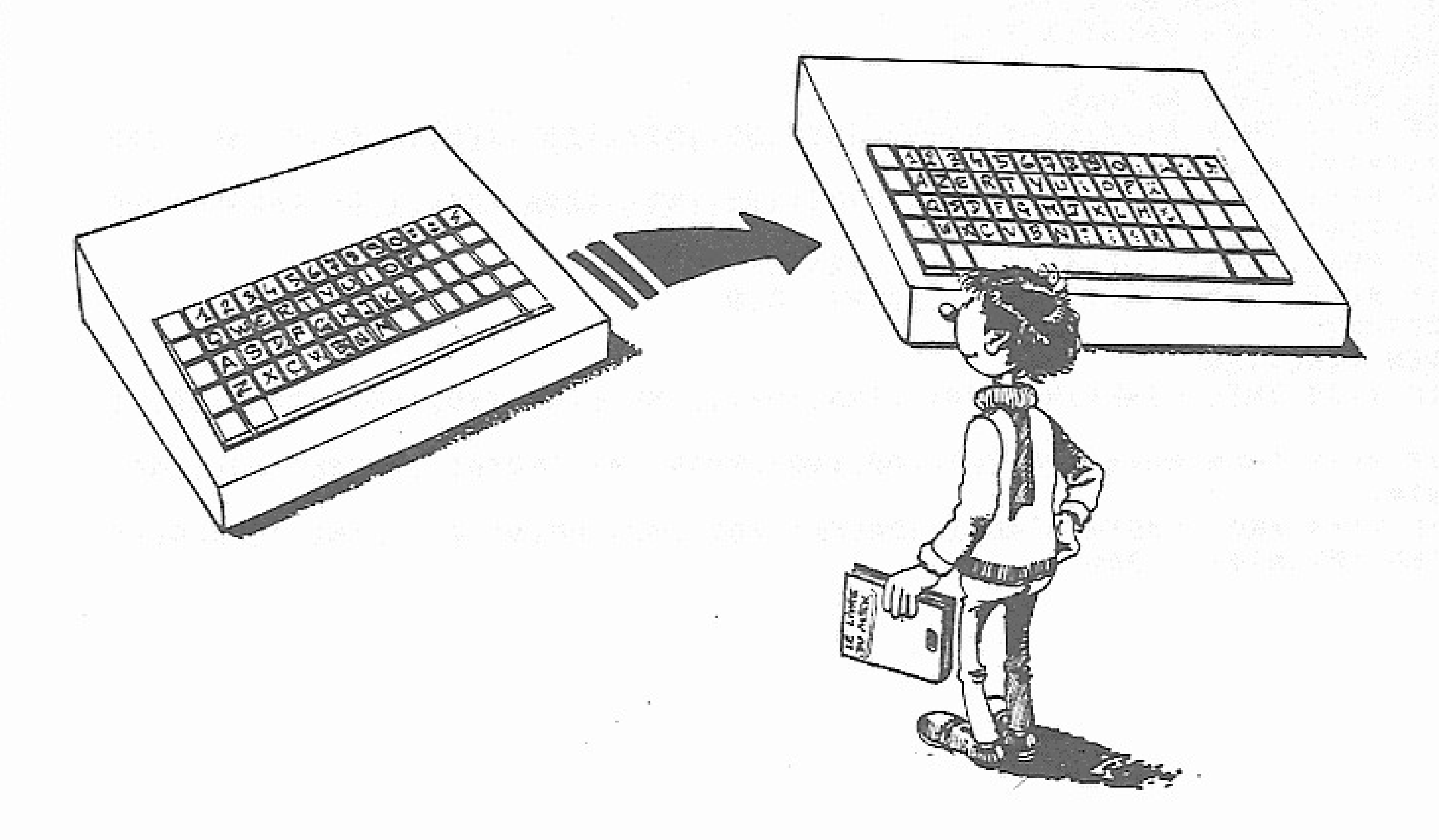
BON AMUSEMENT !!!





```
10 DEFINT A-Z
20 SCREEN 0
30 INPUT"0=CLAVIER , 1=MANETTE ":N
40 SCREEN 2
50 OPEN "GRP:" AS #1
60 SPRITE$(0)=CHR$(24)+CHR$(24)+CHR$(24)+CHR$(231)+CHR$(231)+CHR$(24)
HR$(24)+CHR$(24)
70 X=126 :Y=94
80 GOSUB 430
90 GOSUB 120
100 GOSUB 230
110 GOTO 90
120 A=STICK(N)
130 IF A=1 OR A=2OR A=8 THEN Y=Y-1
140 IF A=4 OR A=5OR A=6 THEN Y=Y+1
150 IF A=2 OR A=3 OR A=4 THEN X=X+1
160 IF A=6 OR A=7 OR A=8 THEN X=X-1
170 IF X<0 THEN X=0
180 IF X>255 THEN X=255
190 IF Y<0 THEN Y=0
200 IF Y>191 THEN Y=191
210 PUT SPRITE0,(X-4,Y-4),15,0
220 RETURN
230 REM LIGNE
240 IF NOT(STRIG(N)) THEN RETURN
250 IF X>191 THEN GOTO 340
260 IF MO=4 THEN PAINT(X,Y),C
270 PSET(X,Y),C
280 IF MO=1 THEN RETURN
290 IF FL=0 THEN X1=X:Y1=Y:FL=1:LINE(195,158)-(235,168),1,BF:PRESET(20
,160):PRINT #1," DES ":RETURN
300 IF FL=1 THEN X2=X:Y2=Y:FL=0:LINE(195,158)-(235,168),1,BF:PRESET(20
,160):PRINT #1," ORG "
310 IF MO=2 THEN LINE(X1,Y1)-(X2,Y2),C
320 IF MO=3 THEN LINE(X1,Y1)-(X2,Y2),C,B
330 RETURN
340 REM FONCTION
350 IF Y<70 THEN LINE(195,178)-(235,188),1,BF:LINE(195,158)-(235,168),
,BF
360 IF Y<15 THEN MO=1 :PRESET(200,180):PRINT #1,"POINT":PRESET(200,160
:PRINT#1,"   `  "
370 IF Y>14 AND Y<30THEN MO=2 :PRESET(200,180):PRINT #1,"LINE ":PRESET
200,160):PRINT#1," ORG "
```

```
380 IF Y>29 AND Y<45 THEN MO=3:PRESET(200,180):PRINT#1," BOX ":PRESET(2
00,160):PRINT #1," ORG "
390 IF Y>44 AND Y<60 THEN MO=4 :PRESET(200,180):PRINT#1,"PAINT":PRESET(
200,160):PRINT #1,"     "
400 IF Y>59 AND Y<75 THEN CLS  :GOSUB 430
410 IF Y>74 AND Y<90 THEN C=C+1:C=C MOD 16:LINE(200,80)-(206,88),C,BF
420 RETURN
430 REM PREPA MENU
440 LINE(4,0)-(255,191),15,B
450 LINE(191,0)-(191,191),15
460 MO=1:FL=0:C=15
470 PRESET(200,5):PRINT #1,"O POINT"
480 PRESET(200,20):PRINT #1,"O LINE "
490 PRESET(200,35):PRINT #1,"O BOX  "
500 PRESET(200,50):PRINT #1,"O PAINT"
510 PRESET(200,65):PRINT #1,"O CLEAR"
520 PRESET(200,80):PRINT #1,"O COLOR"
530 LINE(200,80)-(206,88),C,BF
540 LINE(195,178)-(240,188),1,BF
550 LINE(195,158)-(240,168),1,BF
560 PRESET(200,180):PRINT #1,"POINT"
570 RETURN
```





## 7.11 Moniteur.

### 7.11.1 Généralités.

Ce programme est le plus conséquent du présent manuel et justifie à lui seul son achat.

Voici donc un moniteur complètement écrit en assembleur. Il occupe un peu plus de 1 K.

Ce moniteur rendra de grands services à tous ceux qui travaillent en assembleur.

Pour générer le programme assembleur, on se sert du programme BASIC situé section 7.11.4. Les explications quant à l'utilisation des programmes BASIC et MONITEUR se trouvent ci-dessous.

### 7.11.2 Utilisation du programme BASIC.

1- Encoder le programme BASIC et le lancer. S'il y a des erreurs d'encodage au niveau des DATAS, le programme le signale.

2- Placer une cassette vierge dans l'enregistreur et enfoncer les toucher RECORD et PLAY.

3- Taper ENTER.
   Le programme BASIC créera alors un programme en langage machine appelé MON.

### 7.11.3 Chargement, initialisation et utilisation du programme ASSEMBLEUR.

1- Protéger le haut de mémoire par CLEAR 200,&HEC00

2- Charger le moniteur par BLOAD"MON"

3- L'initialiser par DEFUSR=&HF0D9 : L=USR(0)

Pour lancer le moniteur, il suffit de taper CMD.

Passons en revue les différentes commandes du moniteur :

A adresse1 (adresse2) : visualisation en ASCII d'une partie de mémoire située entre adresse1 et adresse2. Si adresse2 est absente, visualisation de 128 adresses à partir d'adresse1.
EXEMPLE : A1300 1500

CTRL B retour au BASIC.

B:visualisation d'un point d'arrêt.(BREAK POINT)

BR:suppression du point d'arrêt.(BREAK RESET).

BS adresse1 : positionnement du point d'arrêt à l'adresse1.
EXEMPLE : BSA009 positionne le point d'arrêt en A009.

D adresse1 (adresse2) : visualisation en hexadécimal d'une partie de mémoire située entre adresse1 et adresse2. Si adresse2 est absente, visualisation de 128 adresses à partir d'adresse1.
EXEMPLE : D0000 visualise de 0000 à 80H en hexa.

E adresse : passage en mode édition. Affiche le contenu de l'octet à l'adresse spécifiée, permet de le modifier, puis passe automatiquement à l'adresse suivante.
Pour en sortir : CTRL STOP
EXEMPLE : EA000 affiche le contenu de A000

F adresse1 adresse2 octet1octet2 : recherche la suite d'octets octet1octet2 entre adresse1 et adresse2.
EXEMPLE : F0000 7FFF 6677 recherche de l'octet 66 suivi de l'octet 77 entre les adresses 0000 et 7FFF.

G adresse1 : lancement de l'exécution d'un programme assembleur se trouvant à l'adresse1.
EXEMPLE : GA000 lancera l'exécution en A000.

I port : lecture d'un port.
EXEMPLE : I90 : lecture du port 90.

J: exécute le programme situé à l'adresse contenue dans le PC.

M adresse1 adresse2 adresse3 : déplacement du contenu mémoire entre adresse1 et adresse2 vers adresse3.



O port valeur : écriture de valeur dans port.
EXEMPLE : O12 23b écriture de la valeur 23 sur le port n° 12.
REM : ne pas oublier le blanc après la valeur

P : commutation du périphérique de sortie écran/imprimante. cette fonction est une bascule.

X : visualisation des registres primaires. (AF,BC,DE,HL,SP et PC)

X' : visualisation des registres secondaires. (AF',BC',DE',HL',IX et IY).

Xn : modification du registre primaire n ou n vaut : A,F,B,D,H,S ou P.

X'n : modification du registre secondaire n ou n vaut : A,F,B,D,H,X ou Y.

Z adresse1 adresse2 valeur : copie de la valeur dans la zone mémoire comprise entre adresse1 et adresse2.

REMARQUES :

- Toutes les adresses et valeurs se donnent en hexadécimal.
- La commande doit être suivie de l'adresse sans blanc et les adresses doivent être séparées par un blanc.
- Pour une adresse, seuls les 4 chiffres de droite sont pris en compte.
- Pour une valeur, seuls les deux chiffres de droite sont pris en compte.
- Une adresse entre parenthèses est optionnelle.
- Toute commande non reconnue ou erronnée donne lieu au message d'erreur COMMANDE?.
- Les commandes D, A et F peuvent être arrêtées par CTRL STOP
- La visualisation peut être gelée en appuyant sur la barre d'espace.

Utilisation du point d'arrêt.

Un point d'arrêt permet de forcer un retour au moniteur à l'intérieur d'un programme en langage machine.

Un point d'arrêt ne peur être positionné que dans la RAM.

Lors du retour au moniteur, l'affichage des registres primaires est automatique.

EXEMPLE : dans cet exemple, XX remplace toute valeur affichée par l'ordinateur.

1) vider le registre HL :
```
-XH XXXX 0000
```

2) écrire un petit programme :
```
-EA000
A000 XX 21    LD  HL;1234H
A001 XX 34
A002 XX 12
           CTRL STOP
```

3) positionnement du point d'arrêt en A003 :
```
-BSA003
```

4) lancement du programme :
```
-GA000
```
le programme s'exécute et, comme un point d'arrêt à été fixé en A003, il retourne automatiquement au moniteur en affichant le contenu des registres.

REMARQUE : Le listing source du moniteur qui se trouve directement après le programme BASIC à été réalisé par l'assembleur M80 de MICROSOFT en CPM 2.2. Il s'agit du listing .PRN généré par cet assembleur.



7.11.4 Programme BASIC de création du moniteur.

```
10 CLEAR 200,&HEC00
20 CLS
30 PRINT"chargement de la memoire"
40-FOR I=&HEC00 TO &HF0E6
50 READ A$:A=VAL("&H"+A$)
60 POKE I,A:CK=CK+A
70 NEXT
80 IF CK<>153374! THEN PRINT"ERREUR DE DATA ":STOP
90 CLS
100 PRINT"preparez votre cassette et tapez enter"
110 INPUT A$
120 BSAVE"MON",&HEC00,&HF0FF
130 DATA C3,69,EC,C3,E8,EF,00,00,00,00,00,00,0C,53,55,50,45,52,4D,4F
,49,54,45,55,52,20,20
140 DATA 20,31,2E,30,0D,0A,0A,44,2E,20,4D,41,52,54,49,4E,20,28,43,29
,31,39,38,34,0D,0A,0A
150 DATA 00,CD,9F,00,FE,03,CA,85,EC,FE,1B,C4,18,00,FE,41,D8,FE,5B,DC
,5F,C9,CD,9C,00,C8,CD
160 DATA 9F,00,FE,03,C4,9F,00,FE,03,28,26,C9,21,93,F0,CD,7B,52,FB,18
,3E,FF,32,1c,F1,ED,73
170 DATA 1A,F1,22,18,F1,AF,32,06,EC,21,0C,EC,CD,7B,52,21,00,00,22,07
,ED,7B,1A,F1,CD,28,73
180 DATA 3E,2D,DF,CD,39,EC,21,A2,EC,23,BE,38,08,23,5E,23,56,20,F6,EB
,18,BD,02,CE,EC,41,6F
190 DATA ED,42,8B,EF,44,2E,ED,45,8D,ED,46,F0,EE,47,97,EE,49,44,EF,4A
,EE,4D,BA,EE,4F,53,EF
200 DATA 50,6A,EF,58,E7,ED,5A,DC,EE,FF,ED,7B,1A,F1,97,32,1C,F1,E1,2A
,F1,C9,7E,2B,18,05,7C
210 DATA CD,E4,EC,7D,F5,0F,0F,0F,0F,CD,ED,EC,F1,E6,0F,C6,90,27,CE,40
,CD,2C,F0,C9,CD,05,ED
220 DATA DA,60,EC,0C,0D,CA,60,EC,C9,21,00,00,4D,CD,39,EC,FE,0D,C8,FE
,C8,D6,30,D8,C6,E9,D8
230 DATA C6,06,38,03,C6,07,D8,04,C6,0A,5F,16,00,29,29,29,29,19,0C,C3
,ED,CD,F9,EC,01,80,00
240 DATA FE,0D,28,0E,E5,CD,F9,EC,D1,ED,52,DA,60,EC,44,4D,03,EB,F5,3E
,CD,2C,F0,3E,0A,CD,2C
250 DATA F0,F1,CD,DF,EC,CD,4F,EC,3E,20,CD,2C,F0,7E,23,CD,E4,EC,0B,78
,CA,85,EC,7D,E6,07,28
260 DATA D9,18,E6,CD,F9,EC,01,80,00,FE,0D,28,0E,E5,CD,F9,EC,D1,ED,52
,60,EC,44,4D,03,EB,F5
270 DATA 3E,0D,CD,2C,F0,3E,0A,CD,2C,F0,F1,CD,DF,EC,3E,20,CD,2C,F0,CD
,EC,7E,23,FE,20,30,02
280 DATA 3E,2E,F5,3E,20,CD,2C,F0,F1,CD,2C,F0,0B,78,B1,CA,85,EC,7D,E6
,28,CC,18,DE,CD,F9,EC
290 DATA CD,28,73,CD,DF,EC,3E,20,DF,7E,CD,E4,EC,3E,20,DF,E5,CD,05,ED
,85,EC,0C,0D,4D,E1,28
300 DATA 01,71,FE,0D,20,02,23,23,2B,18,D9,CD,39,EC,FE,0D,28,2E,FE,27
,1B,21,85,F0,11,00,00
```

```
310 DATA BE,CA,47,EE,23,BE,CA,4C,EE,23,06,05,BE,28,64,23,10,FA,C3,60,EC
,CD,39,EC,FE,0D,28,10
320 DATA 21,8C,F0,11,0C,00,18,DC,21,43,F0,11,08,F1,18,06,21,64,F0,11,17
,F1,D5,CD,78,66,E1,CD
330 DATA DB,EC,3E,20,DF,CD,DB,EC,06,05,3E,20,DF,CD,08,EC,CD,D8,EC,10,F5
,18,21,21,0B,F1,18,03
340 DATA 21,0A,F1,19,E5,3E,20,DF,7E,CD,E4,EC,3E,20,DF,CD,05,ED,DA,60,EC
,0C,0D,28,03,7D,E1,77
350 DATA C3,85,EC,26,00,68,29,19,11,FF,F0,19,3E,20,DF,56,2B,5E,EB,D5,CD
,DF,EC,3E,20,DF,CD,05
360 DATA ED,DA,60,EC,0C,0D,28,DC,d1,EB,73,23,72,18,d5,7E,23,DF,C9,CD,F9
,EC,22,00,F1,2A,0A,F1
370 DATA 7D,6C,67,E5,F1,ED,5B,06,F1,ED,4B,08,F1,ED,7B,02,F1,2A,04,F1,E5
,2A,00,F1,E3,C9,CD,29
380 DATA EF,03,C5,D5,CD,F9,EC,D1,C1,E5,B7,ED,52,E1,30,06,EB,ED,B0,C3,85
,EC,0B,09,EB,09,03,ED
390 DATA B8,C3,85,EC,CD,29,EF,C5,D5,CD,F9,EC,7D,E1,C1,54,5D,13,77,ED,B0
,C3,85,EC,CD,29,EF,C5
400 DATA D5,CD,F9,EC,44,4D,E1,D1,13,3E,0D,CD,2C,F0,3E,0A,CD,2C,F0,78,BE
,28,09,23,1B,7B,B2,20
410 DATA F6,C3,85,EC,23,1B,79,BE,20,ED,2B,CD,DF,EC,3E,20,CD,2C,F0,CD,4F
,EC,23,18,DE,CD,F9,EC
420 DATA FE,0D,CA,60,EC,E5,CD,F9,EC,D1,FE,0D,CA,60,EC,B7,ED,52,DA,60,EC
,44,4D,C9,CD,F9,EC,CD
430 DATA 28,73,4D,ED,78,CD,E4,EC,C3,85,EC,CD,F9,EC,FE,0D,CA,60,EC,E5,CD
,F9,EC,FE,0D,CA,60,EC
440 DATA C1,ED,69,C3,85,EC,3A,06,EC,FE,00,28,0D,AF,32,06,EC,21,C6,F0,CD
,7B,52,C3,85,EC,3D,32
450 DATA 06,EC,21,B4,F0,CD,7B,52,C3,85,EC,CD,39,EC,FE,53,28,20,FE,52,20
,05,CD,D2,EF,18,05,FE
460 DATA 0D,C2,60,EC,21,A2,F0,CD,7B,52,2A,07,EC,CD,DF,EC,CD,28,73,C3,85
,EC,CD,F9,EC,E5,CD,D2
470 DATA EF,E1,E5,22,07,EC,11,09,EC,01,03,00,ED,B0,E1,36,C3,23,11,03,EC
,73,23,72,18,CE,ED,5B
480 DATA 07,EC,7A,B3,C8,21,09,EC,01,03,00,ED,B0,21,00,00,22,07,EC,C9,32
,0A,F1,22,04,F1,ED,43
490 DATA 08,F1,ED,53,06,F1,F5,E1,7D,32,0B,F1,ED,73,02,F1,DD,22,0E,F1,F0
,22,0C,F1,D9,22,10,F1
500 DATA ED,43,14,F1,ED,53,12,F1,D9,08,32,16,F1,F5,E1,7D,32,17,F1,08,2A
,07,EC,22,00,F1,CD,D2
510 DATA EF,C3,1C,EE,F5,3A,06,EC,FE,00,28,0C,3E,01,32,16,F4,F1,F5,DF,AF
,32,16,F4,F1,DF,C9,0D
520 DATA 0A,46,20,20,41,20,20,42,43,20,20,20,44,45,20,20,20,48,4C,20,20
,20,53,50,20,20,20,50
530 DATA 43,0D,0A,00,0D,0A,46,27,20,41,27,20,42,43,27,20,20,44,45,27,20
,20,48,4C,27,20,20,49
540 DATA 58,20,20,20,49,59,0D,0A,00,46,41,42,44,48,53,50,46,41,42,44,48
,58,59,0A,0D,43,4F,4D
550 DATA 4D,41,4E,44,45,20,3F,0A,0D,00,0A,0D,42,52,45,41,4B,20,53,45,54
,20,41,54,20,3A,20,00
560 DATA 0A,0D,49,4D,50,52,49,4D,41,4E,54,45,20,4F,4E,0A,0D,00,0A,0D,49
,4D,50,52,49,4D,41,4E
570 DATA 54,45,20,4F,46,46,0A,0D,00,21,0D,FE,36,C3,23,36,00,23,36,EC,C3
,28,41
```



```
                                  .Z80
0000'   C3 0069'                          JP      MON
0003'   C3 03E8'                  BRKENT: JP      BRKRET
0006'   00                        LPTFLG: DB      0
0007'   0000                      BRKSTK: DW      0000
0009'   00 00 00                  BRKTMP: DB      0,0,0
002D                              PROMPT  EQU     "-"
000D                              CR      EQU     13
001B                              ESC     EQU     27
000A                              LF      EQU     10
F416                              PRTF    EQU     0F416H
F100                              REGPC   EQU     0F100H
F102                              REGSP   EQU     0F102H
F104                              REGHL   EQU     0F104H
F106                              REGDE   EQU     0F106H
F108                              REGBC   EQU     0F108H
F10A                              REGA    EQU     0F10AH
F10B                              REGF    EQU     0F10BH
F10C                              REGIY   EQU     0F10CH
F10E                              REGIX   EQU     0F10EH
F110                              REGHLP  EQU     0F110H
F112                              REGDEP  EQU     0F112H
F114                              REGBCP  EQU     0F114H
F116                              REGAP   EQU     0F116H
F117                              REGFP   EQU     0F117H
F117                              REGFT   EQU     0F117H
F118                              TXPSAV  EQU     0F118H
F11A                              SAVESP  EQU     0F11AH
F11C                              MONFLG  EQU     0F11CH
0000                              CODE    EQU     0
6678                              STROUT  EQU     06678H
0018                              OUTDO   EQU     18H
009F                              CHGET   EQU     009FH
009C                              CHSNS   EQU     009CH
7328                              CRLF    EQU     07328H
527B                              STRNG   EQU     0527BH
000C'   0C                        CMSG:   DB      12
000D'   53 55 50 45                       DB      'SUPERMONITEUR   1.0',
0011'   52 4D 4F 4E                               ,10
0015'   49 54 45 55
0019'   52 20 20 20
001D'   31 2E 30 0D
0021'   0A 0A
0023'   44 2E 20 4D                       DB      'D. MARTIN (C) 1984',1
0027'   41 52 54 49                               ,10,0
002B'   4E 20 28 43
002F'   29 20 31 39
0033'   38 34 0D 0A
0037'   0A 00
0039'   CD 009F                   CHRIN:  CALL    CHGET
003C'   FE 03                             CP      3        -> CTL C ou CTL
003E'   CA 0085'                          JP      Z,MAIN
0041'   FE 1B                             CP      ESC
0043'   C4 0018                           CALL    NZ,OUTDO
0046'   FE 41                             CP      "A"
0048'   D8                                RET     C
```

```
0049'   FE 5B           CP      "Z"+1
004B'   D0              RET     NC
004C'   E6 5F           AND     95
004E'   C9              RET
004F'   CD 009C   CHKSTP: CALL  CHSNS
0052'   C8              RET     Z
0053'   CD 009F         CALL    CHGET
0056'   FE 03           CP      3
0058'   C4 009F         CALL    NZ,CHGET
005B'   FE 03           CP      3
005D'   28 26           JR      Z,MAIN
005F'   C9              RET
0060'   21 0493'  MONERR: LD    HL,MSGERR
0063'   CD 527B         CALL    STRNG
0066'   FB              EI
0067'   18 1C           JR      MAIN
0069'             MON:
0069'   3E FF           LD      A,255
006B'   32 F11C         LD      (MONFLG),A
006E'   ED 73 F11A      LD      (SAVESP),SP
0072'   22 F118         LD      (TXPSAV),HL
0075'   AF              XOR     A
0076'   32 0006'        LD      (LPTFLG),A
0079'   21 000C'        LD      HL,CMSG
007C'   CD 527B         CALL    STRNG
007F'   21 0000         LD      HL,0
0082'   22 0007'        LD      (BRKSTK),HL
0085'   ED 7B F11A MAIN: LD     SP,(SAVESP)
0089'   CD 7328         CALL    CRLF
008C'   3E 2D           LD      A,PROMPT
008E'   DF              RST     18H
008F'   CD 0039'        CALL    CHRIN
0092'   21 00A2'        LD      HL,COMTAB-1
0095'   23        ONGO1: INC    HL
0096'   BE              CP      (HL)
0097'   38 08           JR      C,BADCOM
0099'   23              INC     HL
009A'   5E              LD      E,(HL)
009B'   23              INC     HL
009C'   56              LD      D,(HL)
009D'   20 F6           JR      NZ,ONGO1
009F'   EB              EX      DE,HL
00A0'   E9              JP      (HL)
00A1'   18 BD     BADCOM: JR    MONERR
00A3'   02        COMTAB: DB    2
00A4'   00CE'           DW      QUIT
00A6'   41              DB      "A"
00A7'   016F'           DW      ASCII
00A9'   42              DB      "B"
00AA'   038B'           DW      BREAK
00AC'   44              DB      "D"
00AD'   012E'           DW      DUMP
00AF'   45              DB      "E"
00B0'   01BD'           DW      MEMORY
00B2'   46              DB      "F"
00B3'   02F0'           DW      FIND
```



```
00B5'   47              DB      "G"
00B6'   0297'           DW      GOEXEC
00B8'   49              DB      "I"
00B9'   0344'           DW      INP
00BB'   4A              DB      'J'
00BC'   029D'           DW      EXEC
00BE'   4D              DB      "M"
00BF'   02BA'           DW      MOVE
00C1'   4F              DB      "O"
00C2'   0353'           DW      OUT
00C4'   50              DB      'P'
00C5'   036A'           DW      LPTR
00C7'   58              DB      "X"
00C8'   01E7'           DW      REG
00CA'   5A              DB      "Z"
00CB'   02DC'           DW      ZERO
00CD'   FF              DB      255
00CE'   ED 7B F11A QUIT: LD     SP,(SAVESP)
00D2'   97              SUB     A
00D3'   32 F11C         LD      (MONFLG),A
00D6'   E1              POP     HL
00D7'   2A F118         LD      HL,(TXPSAV)
00DA'   C9              RET
00DB'   7E        OUTAM: LD     A,(HL)
00DC'   2B              DEC     HL
00DD'   18 05           JR      HEXOUT
00DF'   7C        OUTHL: LD     A,H
00E0'   CD 00E4'        CALL    HEXOUT
00E3'   7D              LD      A,L
00E4'   F5        HEXOUT: PUSH  AF
00E5'   0F              RRCA
00E6'   0F              RRCA
00E7'   0F              RRCA
00E8'   0F              RRCA
00E9'   CD 00ED'        CALL    OUTHEX
00EC'   F1              POP     AF
00ED'   E6 0F     OUTHEX: AND   15
00EF'   C6 90           ADD     A,144
00F1'   27              DAA
00F2'   CE 40           ADC     A,64
00F4'   27              DAA
00F5'   CD 042C'        CALL    CRT
00F8'   C9              RET
00F9'   CD 0105'  HEXGET: CALL  HEXIN
00FC'   DA 0060'        JP      C,MONERR
00FF'   0C              INC     C
0100'   0D              DEC     C
0101'   CA 0060'        JP      Z,MONERR
0104'   C9              RET
0105'   21 0000   HEXIN: LD     HL,CODE
0108'   4D              LD      C,L
0109'   CD 0039'  IN1:  CALL    CHRIN
010C'   FE 0D           CP      13
010E'   C8              RET     Z
010F'   FE 20           CP      32
0111'   C8              RET     Z
```

```
0112'   D6 30              SUB     "0"
0114'   D8                 RET     C
0115'   C6 E9              ADD     A,"0"-"G"
0117'   D8                 RET     C
0118'   C6 06              ADD     A,6
011A'   38 03              JR      C,IN2
011C'   C6 07              ADD     A,7
011E'   D8                 RET     C
011F'   04         IN2:    INC     B
0120'   C6 0A              ADD     A,10
0122'   5F                 LD      E,A
0123'   16 00              LD      D,0
0125'   29                 ADD     HL,HL
0126'   29                 ADD     HL,HL
0127'   29                 ADD     HL,HL
0128'   29                 ADD     HL,HL
0129'   19                 ADD     HL,DE
012A'   0C                 INC     C
012B'   C3 0109'           JP      IN1
012E'   CD 00F9'   DUMP:   CALL    HEXGET
0131'   01 0080            LD      BC,128
0134'   FE 0D              CP      CR
0136'   28 0E              JR      Z,DMPADR
0138'   E5                 PUSH    HL
0139'   CD 00F9'           CALL    HEXGET
013C'   D1                 POP     DE
013D'   ED 52              SBC     HL,DE
013F'   DA 0060'           JP      C,MONERR
0142'   44                 LD      B,H
0143'   4D                 LD      C,L
0144'   03                 INC     BC
0145'   EB                 EX      DE,HL
0146'   F5         DMPADR: PUSH    AF
0147'   3E 0D              LD      A,13
0149'   CD 042C'           CALL    CRT
014C'   3E 0A              LD      A,10
014E'   CD 042C'           CALL    CRT
0151'   F1                 POP     AF
0152'   CD 00DF'           CALL    OUTHL
0155'   CD 004F'   DMPNXT: CALL    CHKSTP
0158'   3E 20              LD      A,20H
015A'   CD 042C'           CALL    CRT
015D'   7E                 LD      A,(HL)
015E'   23                 INC     HL
015F'   CD 00E4'           CALL    HEXOUT
0162'   0B                 DEC     BC
0163'   78                 LD      A,B
0164'   B1                 OR      C
0165'   CA 0085'           JP      Z,MAIN
0168'   7D                 LD      A,L
0169'   E6 07              AND     7
016B'   28 D9              JR      Z,DMPADR
016D'   18 E6              JR      DMPNXT
016F'   CD 00F9'   ASCII:  CALL    HEXGET
0172'   01 0080            LD      BC,128
0175'   FE 0D              CP      CR
```



```
0177'   28 0E              JR      Z,DMPAD2
0179'   E5                 PUSH    HL
017A'   CD 00F9'           CALL    HEXGET
017D'   D1                 POP     DE
017E'   ED 52              SBC     HL,DE
0180'   DA 0060'           JP      C,MONERR
0183'   44                 LD      B,H
0184'   4D                 LD      C,L
0185'   03                 INC     BC
0186'   EB                 EX      DE,HL
0187'   F5         DMPAD2: PUSH    AF
0188'   3E 0D              LD      A,13
018A'   CD 042C'           CALL    CRT
018D'   3E 0A              LD      A,10
018F'   CD 042C'           CALL    CRT
0192'   F1                 POP     AF
0193'   CD 00DF'           CALL    OUTHL
0196'   3E 20              LD      A,32
0198'   CD 042C'           CALL    CRT
019B'              DMPANX:
019B'   CD 004F'           CALL    CHKSTP
019E'   7E                 LD      A,(HL)
019F'   23                 INC     HL
01A0'   FE 20              CP      32
01A2'   30 02              JR      NC,OK
01A4'   3E 2E              LD      A,2EH
01A6'   F5         OK:     PUSH    AF
01A7'   3E 20              LD      A,32
01A9'   CD 042C'           CALL    CRT
01AC'   F1                 POP     AF
01AD'   CD 042C'           CALL    CRT
01B0'   0B                 DEC     BC
01B1'   78                 LD      A,B
01B2'   B1                 OR      C
01B3'   CA 0085'           JP      Z,MAIN
01B6'   7D                 LD      A,L
01B7'   E6 07              AND     7
01B9'   28 CC              JR      Z,DMPAD2
01BB'   18 DE              JR      DMPANX
01BD'   CD 00F9'   MEMORY: CALL    HEXGET
01C0'   CD 7328    MEMNXT: CALL    CRLF
01C3'   CD 00DF'           CALL    OUTHL
01C6'   3E 20              LD      A,20H
01C8'   DF                 RST     18H
01C9'   7E                 LD      A,(HL)
01CA'   CD 00E4'           CALL    HEXOUT
01CD'   3E 20              LD      A,20H
01CF'   DF                 RST     18H
01D0'   E5                 PUSH    HL
01D1'   CD 0105'           CALL    HEXIN
01D4'   DA 0085'           JP      C,MAIN
01D7'   0C                 INC     C
01D8'   0D                 DEC     C
01D9'   4D                 LD      C,L
01DA'   E1                 POP     HL
01DB'   28 01              JR      Z,MEMSKP
```

```
01DD'   71              LD      (HL),C
01DE'   FE 0D   MEMSKP: CP      CR
01E0'   20 02           JR      NZ,MEMDEC
01E2'   23              INC     HL
01E3'   23              INC     HL
01E4'   2B      MEMDEC: DEC     HL
01E5'   18 D9           JR      MEMNXT
01E7'   CD 0039' REG:   CALL    CHRIN
01EA'   FE 0D           CP      CR
01EC'   28 2E           JR      Z,REGALL
01EE'   FE 27           CP      "'"
01F0'   28 1B           JR      Z,REGTAL
01F2'   21 0485'        LD      HL,REGTAB
01F5'   11 0000         LD      DE,CODE
01F8'   BE      CHKREG: CP      (HL)
01F9'   CA 0247'        JP      Z,REGXF
01FC'   23              INC     HL
01FD'   BE              CP      (HL)
01FE'   CA 024C'        JP      Z,REGXA
0201'   23              INC     HL
0202'   06 05           LD      B,5
0204'   BE      CHKLOP: CP      (HL)
0205'   28 64           JR      Z,REGX
0207'   23              INC     HL
0208'   10 FA           DJNZ    CHKLOP
020A'   C3 0060'        JP      MONERR
020D'   CD 0039' REGTAL: CALL   CHRIN
0210'   FE 0D           CP      CR
0212'   28 10           JR      Z,REGALT
0214'   21 048C'        LD      HL,REGTBX
0217'   11 000C         LD      DE,CODE+REGFT-REGF
021A'   18 DC           JR      CHKREG
021C'   21 0443' REGALL: LD     HL,REGTBH
021F'   11 F10B         LD      DE,REGF
0222'   18 06           JR      REGALH
0224'   21 0464' REGALT: LD     HL,REGTBT
0227'   11 F117         LD      DE,REGFT
022A'   D5      REGALH: PUSH    DE
022B'   CD 6678         CALL    STROUT
022E'   E1              POP     HL
022F'   CD 00DB'        CALL    OUTAM
0232'   3E 20           LD      A,20H
0234'   DF              RST     18H
0235'   CD 00DB'        CALL    OUTAM
0238'   06 05           LD      B,5
023A'   3E 20   REGLP2: LD      A,20H
023C'   DF              RST     18H
023D'   CD 00DB'        CALL    OUTAM
0240'   CD 00DB'        CALL    OUTAM
0243'   10 F5           DJNZ    REGLP2
0245'   18 21           JR      JPMAIN
0247'   21 F10B REGXF:  LD      HL,REGF
024A'   18 03           JR      REGX1
024C'   21 F10A REGXA:  LD      HL,REGA
024F'   19      REGX1:  ADD     HL,DE
0250'   E5              PUSH    HL
```



```
0251'   3E 20           LD      A,20H
0253'   DF              RST     18H
0254'   7E              LD      A,(HL)
0255'   CD 00E4'        CALL    HEXOUT
0258'   3E 20           LD      A,20H
025A'   DF              RST     18H
025B'   CD 0105'        CALL    HEXIN
025E'   DA 0060'        JP      C,MONERR
0261'   0C              INC     C
0262'   0D              DEC     C
0263'   28 03           JR      Z,JPMAIN
0265'   7D              LD      A,L
0266'   E1              POP     HL
0267'   77              LD      (HL),A
0268'   C3 0085' JPMAIN: JP     MAIN
026B'   26 00   REGX:   LD      H,0
026D'   68              LD      L,B
026E'   29              ADD     HL,HL
026F'   19              ADD     HL,DE
0270'   11 F0FF         LD      DE,REGPC-1
0273'   19              ADD     HL,DE
0274'   3E 20           LD      A,20H
0276'   DF              RST     18H
0277'   56              LD      D,(HL)
0278'   2B              DEC     HL
0279'   5E              LD      E,(HL)
027A'   EB              EX      DE,HL
027B'   D5              PUSH    DE
027C'   CD 00DF'        CALL    OUTHL
027F'   3E 20           LD      A,20H
0281'   DF              RST     18H
0282'   CD 0105'        CALL    HEXIN
0285'   DA 0060'        JP      C,MONERR
0288'   0C              INC     C
0289'   0D              DEC     C
028A'   28 DC           JR      Z,JPMAIN
028C'   D1              POP     DE
028D'   EB              EX      DE,HL
028E'   73              LD      (HL),E
028F'   23              INC     HL
0290'   72              LD      (HL),D
0291'   18 D5           JR      JPMAIN
0293'   7E      OUTEM:  LD      A,(HL)
0294'   23              INC     HL
0295'   DF              RST     18H
0296'   C9              RET
0297'   CD 00F9' GOEXEC: CALL   HEXGET
029A'   22 F100         LD      (REGPC),HL
029D'   2A F10A EXEC:   LD      HL,(REGA)
02A0'   7D              LD      A,L
02A1'   6C              LD      L,H
02A2'   67              LD      H,A
02A3'   E5              PUSH    HL
02A4'   F1              POP     AF
02A5'   ED 5B F106      LD      DE,(REGDE)
02A9'   ED 4B F108      LD      BC,(REGBC)
```

```
02AD'  ED 7B F102            LD     SP,(REGSP)
02B1'  2A F104               LD     HL,(REGHL)
02B4'  E5                    PUSH   HL
02B5'  2A F100               LD     HL,(REGPC)
02B8'  E3                    EX     (SP),HL
02B9'  C9                    RET
02BA'  CD 0329'     MOVE:    CALL   GET2
02BD'  03                    INC    BC
02BE'  C5                    PUSH   BC
02BF'  D5                    PUSH   DE
02C0'  CD 00F9'              CALL   HEXGET
02C3'  D1                    POP    DE
02C4'  C1                    POP    BC
02C5'  E5                    PUSH   HL
02C6'  B7                    OR     A
02C7'  ED 52                 SBC    HL,DE
02C9'  E1                    POP    HL
02CA'  30 06                 JR     NC,STD
02CC'  EB                    EX     DE,HL
02CD'  ED B0                 LDIR
02CF'  C3 0085'              JP     MAIN
02D2'  0B           STD:     DEC    BC
02D3'  09                    ADD    HL,BC
02D4'  EB                    EX     DE,HL
02D5'  09                    ADD    HL,BC
02D6'  03                    INC    BC
02D7'  ED B8                 LDDR
02D9'  C3 0085'              JP     MAIN
02DC'  CD 0329'     ZERO:    CALL   GET2
02DF'  C5                    PUSH   BC
02E0'  D5                    PUSH   DE
02E1'  CD 00F9'              CALL   HEXGET
02E4'  7D                    LD     A,L
02E5'  E1                    POP    HL
02E6'  C1                    POP    BC
02E7'  54                    LD     D,H
02E8'  5D                    LD     E,L
02E9'  13                    INC    DE
02EA'  77                    LD     (HL),A
02EB'  ED B0                 LDIR
02ED'  C3 0085'              JP     MAIN
02F0'  CD 0329'     FIND:    CALL   GET2
02F3'  C5                    PUSH   BC
02F4'  D5                    PUSH   DE
02F5'  CD 00F9'              CALL   HEXGET
02F8'  44                    LD     B,H
02F9'  4D                    LD     C,L
02FA'  E1                    POP    HL
02FB'  D1                    POP    DE
02FC'  13                    INC    DE
02FD'  3E 0D                 LD     A,13
02FF'  CD 042C'              CALL   CRT
0302'  3E 0A                 LD     A,10
0304'  CD 042C'              CALL   CRT
0307'  78           TEST1:   LD     A,B
0308'  BE                    CP     (HL)
```



```
0309'  28 09                 JR     Z,TEST2
030B'  23                    INC    HL
030C'  1B                    DEC    DE
030D'  7B                    LD     A,E
030E'  B2                    OR     D
030F'  20 F6                 JR     NZ,TEST1
0311'  C3 0085'              JP     MAIN
0314'  23           TEST2:   INC    HL
0315'  1B                    DEC    DE
0316'  79                    LD     A,C
0317'  BE                    CP     (HL)
0318'  20 ED                 JR     NZ,TEST1
031A'  2B                    DEC    HL
031B'  CD 00DF'              CALL   OUTHL
031E'  3E 20                 LD     A,32
0320'  CD 042C'              CALL   CRT
0323'  CD 004F'              CALL   CHKSTP
0326'  23                    INC    HL
0327'  18 DE                 JR     TEST1
0329'  CD 00F9'     GET2:    CALL   HEXGET
032C'  FE 0D                 CP     CR
032E'  CA 0060'              JP     Z,MONERR
0331'  E5                    PUSH   HL
0332'  CD 00F9'              CALL   HEXGET
0335'  D1                    POP    DE
0336'  FE 0D                 CP     CR
0338'  CA 0060'              JP     Z,MONERR
033B'  B7                    OR     A
033C'  ED 52                 SBC    HL,DE
033E'  DA 0060'              JP     C,MONERR
0341'  44                    LD     B,H
0342'  4D                    LD     C,L
0343'  C9                    RET
0344'  CD 00F9'     INP:     CALL   HEXGET
0347'  CD 7328               CALL   CRLF
034A'  4D                    LD     C,L
034B'  ED 78                 IN     A,(C)
034D'  CD 00E4'              CALL   HEXOUT
0350'  C3 0085'              JP     MAIN
0353'  CD 00F9'     OUT:     CALL   HEXGET
0356'  FE 0D                 CP     CR
0358'  CA 0060'              JP     Z,MONERR
035B'  E5                    PUSH   HL
035C'  CD 00F9'              CALL   HEXGET
035F'  FE 0D                 CP     CR
0361'  CA 0060'              JP     Z,MONERR
0364'  C1                    POP    BC
0365'  ED 69                 OUT    (C),L
0367'  C3 0085'              JP     MAIN
036A'  3A 0006'     LPTR:    LD     A,(LPTFLG)
036D'  FE 00                 CP     0
036F'  28 0D                 JR     Z,SET
0371'  AF                    XOR    A
0372'  32 0006'              LD     (LPTFLG),A
0375'  21 04C6'              LD     HL,PRTRES
0378'  CD 527B               CALL   STRNG
```

```
037B'   C3 0085'              JP     MAIN
037E'   3D            SET:    DEC    A
037F'   32 0006'              LD     (LPTFLG),A
0382'   21 04B4'              LD     HL,PRTSET
0385'   CD 527B               CALL   STRNG
0388'   C3 0085'              JP     MAIN
038B'                 BREAK:
038B'   CD 0039'              CALL   CHRIN
038E'   FE 53                 CP     'S'
0390'   28 20                 JR     Z,BRKSET
0392'   FE 52                 CP     'R'
0394'   20 05                 JR     NZ,NOTRES
0396'   CD 03D2'              CALL   BRKRES
0399'   18 05                 JR     BRKDSP
039B'   FE 0D         NOTRES: CP     CR
039D'   C2 0060'              JP     NZ,MONERR
03A0'   21 04A2'      BRKDSP: LD     HL,MESBRK
03A3'   CD 527B               CALL   STRNG
03A6'   2A 0007'              LD     HL,(BRKSTK)
03A9'   CD 00DF'              CALL   OUTHL
03AC'   CD 7328               CALL   CRLF
03AF'   C3 0085'              JP     MAIN
03B2'                 BRKSET:
03B2'   CD 00F9'              CALL   HEXGET
03B5'   E5                    PUSH   HL
03B6'   CD 03D2'              CALL   BRKRES
03B9'   E1                    POP    HL
03BA'   E5                    PUSH   HL
03BB'   22 0007'              LD     (BRKSTK),HL
03BE'   11 0009'              LD     DE,BRKTMP
03C1'   01 0003               LD     BC,3
03C4'   ED B0                 LDIR
03C6'   E1                    POP    HL
03C7'   36 C3                 LD     (HL),0C3H
03C9'   23                    INC    HL
03CA'   11 0003'              LD     DE,BRKENT
03CD'   73                    LD     (HL),E
03CE'   23                    INC    HL
03CF'   72                    LD     (HL),D
03D0'   18 CE                 JR     BRKDSP
03D2'                 BRKRES:
03D2'   ED 5B 0007'           LD     DE,(BRKSTK)
03D6'   7A                    LD     A,D
03D7'   B3                    OR     E
03D8'   C8                    RET    Z
03D9'   21 0009'              LD     HL,BRKTMP
03DC'   01 0003               LD     BC,3
03DF'   ED B0                 LDIR
03E1'   21 0000               LD     HL,0
03E4'   22 0007'              LD     (BRKSTK),HL
03E7'   C9                    RET
03E8'                 BRKRET:
03E8'   32 F10A               LD     (REGA),A
03EB'   22 F104               LD     (REGHL),HL
03EE'   ED 43 F108            LD     (REGBC),BC
03F2'   ED 53 F106            LD     (REGDE),DE
```



```
03F6'   F5                    PUSH   AF
03F7'   E1                    POP    HL
03F8'   7D                    LD     A,L
03F9'   32 F10B               LD     (REGF),A
03FC'   ED 73 F102            LD     (REGSP),SP
0400'   DD 22 F10E            LD     (REGIX),IX
0404'   FD 22 F10C            LD     (REGIY),IY
0408'   D9                    EXX
0409'   22 F110               LD     (REGHLP),HL
040C'   ED 43 F114            LD     (REGBCP),BC
0410'   ED 53 F112            LD     (REGDEP),DE
0414'   D9                    EXX
0415'   08                    EX     AF,AF'
0416'   32 F116               LD     (REGAP),A
0419'   F5                    PUSH   AF
041A'   E1                    POP    HL
041B'   7D                    LD     A,L
041C'   32 F117               LD     (REGFP),A
041F'   08                    EX     AF,AF'
0420'   2A 0007'              LD     HL,(BRKSTK)
0423'   22 F100               LD     (REGPC),HL
0426'   CD 03D2'              CALL   BRKRES
0429'   C3 021C'              JP     REGALL
042C'   F5            CRT:    PUSH   AF
042D'   3A 0006'              LD     A,(LPTFLG)
0430'   FE 00                 CP     0
0432'   28 0C                 JR     Z,CRTONL
0434'   3E 01                 LD     A,1
0436'   32 F416               LD     (PRTF),A
0439'   F1                    POP    AF
043A'   F5                    PUSH   AF
043B'   DF                    RST    18H
043C'   AF                    XOR    A
043D'   32 F416               LD     (PRTF),A
0440'   F1            CRTONL: POP    AF
0441'   DF                    RST    18H
0442'   C9                    RET
0443'   0D 0A         REGTBH: DB     13,10
0445'   46 20 20 41           DB     "F  A  BC   DE   HL
0449'   20 20 42 43                  PC"
044D'   20 20 20 44
0451'   45 20 20 20
0455'   48 4C 20 20
0459'   20 53 50 20
045D'   20 20 50 43
0461'   0D 0A 00              DB     13,10,0
0464'   0D 0A         REGTBT: DB     13,10
0466'   46 27 20 41           DB     "F' A' BC'  DE'  HL'
046A'   27 20 42 43                  IY"
046E'   27 20 20 44
0472'   45 27 20 20
0476'   48 4C 27 20
047A'   20 49 58 20
047E'   20 20 49 59
0482'   0D 0A 00              DB     13,10,0
0485'   46 41 42 44   REGTAB: DB     "FABDHSP"
```

```
0489'   48 53 50
048C'   46 41 42 44      REGTBX: DB      "FABDHXY"
0490'   48 58 59
0493'   0A 0D 43 4F      MSGERR: DB      10,13,'COMMANDE ?',10,13
0497'   4D 4D 41 4E
049B'   44 45 20 3F
049F'   0A 0D
04A1'   00                       DB      0
04A2'   0A 0D 42 52      MESBRK: DB      10,13,'BREAK SET AT : ',0
04A6'   45 41 4B 20
04AA'   53 45 54 20
04AE'   41 54 20 3A
04B2'   20 00
04B4'   0A 0D 49 4D      PRTSET: DB      10,13,'IMPRIMANTE ON',10
04B8'   50 52 49 4D                      13,0
04BC'   41 4E 54 45
04C0'   20 4F 4E 0A
04C4'   0D 00
04C6'   0A 0D 49 4D      PRTRES: DB      10,13,'IMPRIMANTE OFF',10
04CA'   50 52 49 4D                      13,0
04CE'   41 4E 54 45
04D2'   20 4F 46 46
04D6'   0A 0D 00
04D9'   21 FE0D          DEPART: LD      HL,0FE0DH         ; HOOK CMD
04DC'   36 C3                    LD      (HL),0C3H
04DE'   23                       INC     HL
04DF'   36 00                    LD      (HL),0
04E1'   23                       INC     HL
04E2'   36 EC                    LD      (HL),0ECH
04E4'   C3 4128                  JP      04128H
                                 END
```



7.12 Générateur de caractères.

Ce programme permet de reprogrammer aisément un caractère quelconque dont le code ASCII est compris entre 20H (32) 0FFH (255)

En outre, il permet de sauver un jeu de caractères sur cassette et de le recharger.

L'utilisation est très simple. Il suffit de se déplacer s la matrice 8X8 avec les 4 flèches du clavier.

La barre d'espace permet d'inverser l'état d'un point (allumé - éteint).

La touche CLS permet d'effacer complètement un caractère.

La touche SELECT permet de sortir du mode édition en enregistrant le nouveau caractère.
Après appui sur la touche SELECT, un petit menu est affiché. Il propose de passer à un autre caractère, de sa le set de caractère sur cassette ou de charger la VIDEORA avec le nouveau jeu de caractères.

Enfin, ce programme utilise plusieurs techniques déjà analysées dans ce livre ; à savoir :

a) positionnement du CAPS LOCK.

b) lecture de la touche SELECT.

c) transfert de la VIDEORAM vers la RAM.

d) etc......

Programme BASIC GENCAR.

```
10 CLEAR 200,&HE800
20 DEFINT A-Z
30 DIM M(7,7)
40 GOSUB 1070 : REM INIT USR
50 CLS
60 OUT &HAB,12 : POKE &HFCAB,255
70 PRINT"       GENERATEUR DE CARACTERES"
80 PRINT:PRINT
90 INPUT"CHARGEMENT DE CASSETTE O/N";RP$
100 IF LEFT$(RP$,1)="O" THEN GOSUB 1230
110 INPUT"(A)SCII ou (C)aractere A/C";RP$
120 RP$=LEFT$(RP$,1)
130 IF RP$="C" THEN GOTO 180
140 IF RP$<>"A" THEN GOTO 80
150 PRINT:INPUT"Entrez le code du caractere ";C
160 C$=CHR$(C)
170 GOTO 210
180 PRINT:INPUT"Entrez le caractere a modifier ";C$
190 IF LEN(C$)=1 THEN C=ASC(C$)
200 IF LEN(C$)>1 THEN GOTO 180
210 PRINT : PRINT"Caractere =";C$
220 PRINT:PRINT
230 INPUT"OK O/N ";RP$
240 IF LEFT$(RP$,1)<>"O" THEN GOTO 50
250 GOSUB 870
260 SCREEN 2
270 FOR I=0 TO 8
280 LINE (10+I*10,10)-(10+I*10,90),15
290 LINE (10,10+I*10)-(90,10+I*10),15
300 NEXT I
310 REM SPRITE
320 SP$=CHR$(0)
330 FOR I=1 TO 7
340 SP$=SP$+CHR$(127)
350 NEXT I
360 SPRITE$(0)=SP$
370 GOSUB 1000
380 X=0:Y=0
390 REM MOUVEMENT DU SPRITE
400 PUT SPRITE 1,(11+X*10,10+Y*10),8,0
410 A=STICK(0)
420 IF A=1 AND Y>0 THEN Y=Y-1
430 IF A=3 AND X<7 THEN X=X+1
440 IF A=5 AND Y<7 THEN Y=Y+1
450 IF A=7 AND X>0 THEN X=X-1
460 A=STRIG(0)
470 IF A=-1 THEN M(X,Y)=NOT(M(X,Y)) ELSE GOTO 500
480 IF M(X,Y)=0 THEN LINE(11+X*10,11+Y*10)-(19+X*10,19+Y*10),4,B
490 IF M(X,Y)=-1THEN LINE(11+X*10,11+Y*10)-(19+X*10,19+Y*10),15,
500 REM LECTURE TOUCHE CLS
510 OUT &HAA,8:V=INP(&HA9) AND 2
520 IF V<>0 THEN GOTO 590
530 FOR I=0 TO 7
540 FOR J=0 TO 7
550 M(I,J)=0
560  LINE(11+I*10,11+J*10)-(19+I*10,19+J*10),4,BF
570 NEXT J,I
580 GOTO 390
590 REM TEST SELECT
600 OUT &HAA,7:V=INP(&HA9) AND 64
610 IF V<>0 THEN GOTO 860
620 AD=&HE800+8*C
630 FOR I=0 TO 7
640 V$=""
650 FOR J=0 TO 7
660 V$=V$+RIGHT$(STR$(ABS(M(J,I))),1)
670 NEXT J
680 PRINTV$
690 POKE AD+I,VAL("&B"+V$)
700 NEXT I
710 SCREEN 0
720 CLS
730 PRINT"1 - AUTRE CARACTERE "
740 PRINT
750 PRINT"2 - SAUVEGARDE SUR CASSETTE "
760 PRINT
770 PRINT"3 - COPIE DANS VIDEORAM"
780 PRINT:PRINT
790 INPUT"VOTRE CHOIX ";CH
800 IF CH=1 THEN GOTO 50
810 IF CH=2 THEN GOSUB1190
820 IF CH<>3 THEN GOTO 720
830 L=USR1(BASE(2))
840  CLS
```

```
850 STOP
860 GOTO 390
870 REM LECTURE DE TGP
880 AD=&HE800+C*8
890 FOR I=0 TO 7
900 VL=PEEK(AD+I)
910 VL$=BIN$(VL)
920 VL$="00000000"+VL$
930 VL$=RIGHT$(VL$,8)
940 FOR J=0 TO 7
950 VV$=MID$(VL$,J+1,1)
960 M(J,I)=-VAL(VV$)
970 NEXT J
980 NEXT I
990 RETURN
1000 REM AFFICHAGE
1010 FOR I=0 TO 7
1020 FOR J=0 TO 7
1030 IF M(I,J)=0 THEN LINE(11+I*10,11+J*10)-(19+I*10,19+J*10),4,BF
1040 IF M(I,J)=-1THEN LINE(11+I*10,11+J*10)-(19+I*10,19+J*10),15,BF
1050 NEXT J,I
1060 RETURN
1070 REM INIT USR
1080 FOR I=&HF000 TO &HF01A
1090 READ A$
1100 POKE I,VAL("&H"+A$)
1110 NEXT I
1120 DEFUSR0=&HF000
1130 DEFUSR1=&HF00D
1140 REM LECTURE TGP
1150 VA=BASE(2)
1160 L=USR0(VA)
1170 RETURN
1180 DATA CD,8A,2F,11,00,E8,01,00,08,CD,59,00,C9,CD,8A,2F,11,00,E8,01,0
0,08,EB,CD,5C,00,C9
1190 REM ECRITURE CASSETTE
1200 INPUT"NOM ";NM$
1210 BSAVE NM$,&HE800,&HEFFF
1220 RETURN
1230 REM LECTURE CASSETTE
1240 INPUT"NOM ";NM$
1250 BLOAD NM$
1260 RETURN
```



## 7.13 Programme : programmation des sprites.

Pour terminer, nous allons essayer de déplacer un bicolore sur l'écran.

Le mobile sera représenté par une grille de 16 X 16 poin

Nous le programmerons donc sous la forme d'un SPRITE.

Hélas, un sprite ne peut utiliser qu'une seule couleur.

La difficulé est facile à contourner. Il suffit de prog deux sprites et de les afficher aux mêmes coordonnées.

Nous allons déplacer une soucoupe rouge et verte sur u blanc.

Remplaçons les points à allumer par 1 et les transparents par 0. Nous obtenons ceci :

SPRITE rouge

```
0000000000000000
0000011001100000          **  **
0000001001000000           *  *
0000000110000000            **
0000001111000000           ****
0000010000100000          *    *
0000111001110000         ***  ***
0001111111111000        **********
0001111111111000        **********
0001100000011000        **      **
0001111111111000        **********
0001111111111000        **********
0001000110001000        *   **   *
0001000110001000        *   **   *
0010101111010100       * * **** * *
0010100000010100       * *      * *
```

SPRITE vert

```
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000001111000000          ****
0000000110000000           **
0000000000000000
0000000000000000
0000011111100000         ******
0000000000000000
0000000000000000
0000010000100000         *    *
```

Décomposons chaque dessin en 32 octets. Les 16 premiers sont formés par les 16 lignes en ne considérant que les 8 points de gauche, les 16 autres sont formés par les 16 lignes en ne considérant que les 8 points de droite.

A) Réalisation de la programmation du problème en BASIC avec les instructions standard.

```
10  STOP ON
20  ON STOP GOSUB 900
30  COLOR 15,15,15        ;REM FOND BLANC
40  SCREEN 2,2            ;REM GRAPHIQUE M 2 , SPRITE 16 x 16
50  FOR I=1 TO 32         ;REM LECTURE SPRITE 1
60  READ A$
70  S$=S$+CHR$(VAL("&B"+A$))
80  NEXT I
90  SPRITE$(1)=S$
100 FOR I=1 TO 32         ;REM LECTURE SPRITE 2
110 READ A$
120 T$=T$+CHR$(VAL("&B"+A$))
130 NEXT I
140 SPRITE$(2)=T$
150 PUT SPRITE 0,(X,20),8,1 ;REM AFFICHE SPRITE 1 EN ROUGE
160 PUT SPRITE 1,(X,20),3,2 ;REM AFFICHE SPRITE 2 EN VERT
170 X=X+1                   ;REM ON DEPLACE EN HORIZONTALE
180 IF X=256 THEN X=0
190 GOTO 150
200 REM DATA DEMI SPRITE 1 GAUCHE
210 DATA 00000000
220 DATA 00000110
230 DATA 00000010
240 DATA 00000001
```



```
250 DATA 00000011
260 DATA 00000100
270 DATA 00001110
280 DATA 00011111
290 DATA 00011111
300 DATA 00011000
310 DATA 00011111
320 DATA 00011111
330 DATA 00010001
340 DATA 00010001
350 DATA 00101011
360 DATA 00101000
370 REM DATA DEMI SPRITE 1 DROIT
380 DATA 00000000
390 DATA 01100000
400 DATA 01000000
410 DATA 10000000
420 DATA 11000000
430 DATA 00100000
440 DATA 01110000
450 DATA 11111000
460 DATA 11111000
470 DATA 00011000
480 DATA 11111000
490 DATA 11111000
500 DATA 10001000
510 DATA 10001000
520 DATA 11010100
530 DATA 00010100
540 REM DATA DEMI SPRITE 2 GAUCHE
550 DATA 00000000
560 DATA 00000000
570 DATA 00000000
580 DATA 00000000
590 DATA 00000000
600 DATA 00000011
610 DATA 00000001
620 DATA 00000000
630 DATA 00000000
640 DATA 00000111
650 DATA 00000000
660 DATA 00000000
670 DATA 00000100
680 DATA 00000000
690 DATA 00000000
700 DATA 00000000
710 REM DATA DEMI SPRITE 2 DROIT
720 DATA 00000000
730 DATA 00000000
740 DATA 00000000
750 DATA 00000000
760 DATA 00000000
```

```
770 DATA 11000000
780 DATA 10000000
790 DATA 00000000
800 DATA 00000000
810 DATA 11100000
820 DATA 00000000
830 DATA 00000000
840 DATA 00100000
850 DATA 00000000
860 DATA 00000000
870 DATA 00000000
900 COLOR 15,1,15
```

Remarque : Les lignes 20 à 60 et 120 à 160 utilisent des variables différentes (S$ et T$). C'est obligatoire car le système établit une relation entre la variable et le SPRITE.

Programmation du problème en langage machine.

1° Ecrivons la valeur de chacun des octets en hexadécimal (ligne 1100 à 1850 du programme BASIC).

Nous obtenons donc les 64 valeurs suivantes :

```
00,06,02,01,03,04,0E,1F,1F,18,1F,1F,11,11,2B,28
00,60,40,80,C0,20,70,F8,F8,18,F8,F8,18,18,D4,14
00,00,00,00,00,03,01,00,00,07,00,00,04,00,00,00
00,00,00,00,00,C0,80,00,00,E0,00,00,20,00,00,00
```

2° Détermination de la valeur des registres.

R0=2 mode graphique 2.

R1=11100010 image affichée, interruption permise.
=E2H mode graphique 2.
Sprite de 16 X 16, facteur 1.

R2=6 TNP de 1800H à 1BFFH.

R3=255 TC de 2000H à 37FFH.

R4=3 TGP de 0000 à 17FFH.

R5=54 (36H) TAS en 1B00H.

R6=7 TGS de 3800H à 3FFFH.



3° Réalisation du programme en assembleur.

A) Sélection du mode d'affichage des SPRITES (registre

Il suffit de mettre la valeur E2H déterminée au point 2
l'adresse 0F3E0H (valeur de VDP(1) ).

```
F000  F3           DI              PAS D'INTERRUPT
F001  3E E2        LD    A,0E2H
F003  32 E0 F3     LD    (0F3E0),A
```

B) Appel du mode SCREEN 2 (graphique 2).

Il suffit de faire un CALL en 0072H.

```
F006  CD 72 00     CALL  0072H     MODE SCREEN 2
```

C) Chargement de la TGS.

On doit charger la TGS avec le dessin des 2 SPRITES don
64 valeurs ont été déterminées au point 1°.

Supposons que nous les avons écrites à l'adresse F100H
jusqu'à l'adresse F13FH.

Un chargement des registres HL,DE et BC et un CALL en 0
suffisent.

```
F009  21 00 F1     LD    HL,0F100H HL pointe sur
F00C  11 00 38     LD    DE,03800H DE pointe sur
F00F  01 40 00     LD    BC,0040H  BC = 64 (COMPT
F012  CD 5C 00     CALL  005CH     TRANSFERT
```

D) Chargement de la TAS

On doit charger la TAS avec 4 octets par SPRITE qui
déterminent sa position, son numéro et sa couleur.

Supposons les 8 valeurs écrites à l'adresse 0F140H.

Une routine identique à la précédente est utilisée.

```
F015  21 40 F1     LD    HL,0F140H HL pointe sur
F018  11 00 1B     LD    DE,01B00H DE pointe sur
F01B  01 08 00     LD    BC,8      BC = compteur
F01E  CD 5C 00     CALL  005CH     TRANSFERT
```

Les 8 valeurs sont les suivantes :

20 (position verticale SPRITE 1)
00 (position horizontale SPRITE 1)
00 (numéro du SPRITE dans TGS)
08 (couleur rouge)

20 (position verticale du SPRITE 2)
00 (position horizontale du SPRITE 2)
01 (muméro du SPRITE dans TGS)
03 (couleur verte)

E) Déplacement du SPRITE ( des SPRITES ).

Il suffit de modifier la coordonnée horizontale des 2 SPRITES (adresse 1B01H et 1B05H).

```
F021  3E 00                LD    A,0        Coordonnée départ
F023  21 01 1B   BOUCLE    LD    HL,01B01H  TAS SPRITE 1
F026  CD 4D 00             CALL  004DH      Ecriture
F029  21 05 1B             LD    HL,01B05H  TAS SPRITE 2
F02C  CD 4D 00             CALL  004DH      Ecriture
F02F  F5                   PUSH  AF         SAUVE A
F030  01 00 30             LD    BC,0300H   BC=DELAI
F033  0B         DELAI     DEC   BC
F034  78                   LD    A,B
F035  FE 00                CP    0
F037  20 FA                JR    NZ,DELAI
F039  F1                   POP   AF         Récupère A
F03A  3C                   INC   A          Incrémente horiz
F03B  18 E6                JR    BOUCLE
```

Pour accélérer ou ralentir le mobile, vous pouvez jouer sur la valeur de BC à l'adresse F031H-F032H.

Remarque : une fois ce programme lancé, il ne peut être interrompu que par RESET ou extinction de l'ordinateur.



Enfin, le programme BASIC qui reprend toute la programm
en assembleur.

```
10 CLEAR 200,&HF000
20 FOR I=&HF100 TO &HF147
30 READ A$ : POKE I,VAL("&H"+A$)
40 NEXT I
50 REM DATA POUR LES SPRITES  (TGS)
60 DATA 00,06,02,01,03,04,0E,1F,1F,18,1F,1F,11,11,2B,28
62 DATA 00,60,40,80,C0,20,70,F8,F8,18,F8,F8,18,18,D4,14
64 DATA 00,00,00,00,00,03,01,00,00,07,00,00,04,00,00,0C
66 DATA 00,00,00,00,00,C0,80,00,00,E0,00,00,20,00,00,0C
67 REM DATA TAS
68 DATA 20,00,00,08,20,00,01,03
70 REM PROGRAMME
75 FOR I=&HF000 TO &HF03C
80 READ A$ : POKE I,VAL("&H"+A$) : NEXT I
85 DEFUSR=&HF000
90 L=USR(0) : REM LANCEMENT DU PROGRAMME
95 DATA F3,3E,E2,32,E0,F3,CD,72,00,21,00,F1,11,00,38,01
96 DATA 40,00,CD,5C,00,21,40,F1,11,00,1B,01,08,00,CD,5C
97 DATA 00,3E,00,21,01,1B,CD,4D,00,21,05,1B,CD,4D,00,F5
98 DATA 01,00,30,0B,78,FE,00,20,FA,F1,3C,18,E6
```

Cette réalisation nous a permis de faire une synthèse sur la programmation des SPRITES, sur les TABLES du VDP sur les différents vecteurs BIOS et sur le déplacement des mobiles.

Vous avez fait vos premiers pas vers la programmation d
jeux complexes.

ATTENTION : CE PROGRAMME EST RESERVE AUX POSSESSEURS D'UN DISQUE.

## 7.14 Passage de variables entre deux programmes BASIC

A chaque RUN d'un nouveau programme, toutes les variables utilisées dans le programme précédent sont remises à 0. Il y a des cas où l'on aimerait que les variables ne soient pas effacées par le changement de programme. Les possesseurs d'un système disque peuvent toujours sauver les variables dans un fichier et les récupérer dans le programme suivant, mais cette technique est longue et lourde.

Le système proposé ici est rapide et facile à mettre en oeuvre. Vous pouvez donc diviser votre application en petits programmes. Les petits programmes vous laissent plus de place pour les variables.

La routine de passage de variables occupe 2 lignes de BASIC dans le programme qui passe les variables et 1 ligne dans le programme récepteur.

Technique :

Les variables sont sauvées juste à la fin du texte du programme (TIP), des pointeurs dans la région de communication gardent en mémoire l'adresse de début des tables.

OF6C2H = adresse de départ de la table des variables simples.
OF6C4H = adresse de départ de la table des variables tableaux.
OF6C6H = adresse de départ de la table de travail des chaînes.

La technique utilisée consiste à fixer l'adresse de ces tables au début du programme qui passe les variables. Evidemment l'adresse choisie doit être supérieure à l'adresse de fin du plus grand des programmes. Il faut prendre l'adresse de fin et ajouter 300 octets environ pour permettre des petites modifications de dernière minute.

Pour connaître la taille de votre programme, notez le nombre d'octets libres (BYTES FREE) lors de l'initialisation. Après l'encodage de votre programme et avant de le lancer , tapez FRE(0) , vous connaitrez la taille de mémoire restante. Soustrayez ce chiffre du nombre d'octets libres et vous obtenez la taille du programme.

L'adresse de fin se détermine en ajoutant la taille du programme à l'adresse de début de programme



Exemple :

```
adresse de début 8001H    32769
octets libres             21999 (dépend de la configurati
FRE(0)                    16897

-> taille du programme  21999-16897 = 5102 octets
-> taille + 300 octets de sécurité  = 5402 octets
-> adresse   32769 + 5402 = 38171   = 951BH
```

Une première ligne doit être exécutée avant toute décla de variables (ligne 50000).

La deuxième ligne (50100) doit être exécutée juste avant du programme récepteur.

La technique utilisée dans ces 2 lignes est assez partic

Dans la ligne 50000, la variable A$ est remplie avec l'adresse déterminée par le calcul ci-dessus (DANS LE PRO BASIC D'EXEMPLE, CETTE VALEUR VAUT A000H) à l'aide d'une fo MKI$, ensuite une variable AN$ est remplie avec 6 cara bidons, le VARPTR de cette variable est modifié pour pointer première adresse de la première table (OF6C2H). Les 6 ad consécutives sont modifiées en une seule manoeuvre à l'aide fonction LSET.

Dans la ligne 50100, la longueur de la variable AN modifiée dynamiquement pour contenir les 104 octets qui s l'adresse OF674H. Cette variable est ensuite copiée dans A$.

Le programme récepteur ne contient qu'une ligne (50200 doit être appelée avant toute déclaration.

La ligne 50200 rétablit l'état des 104 octets sauvés p ligne 50100 du programme précédent.

Remarque : le passage de variables peut s'enchaîner d'u gramme à l'autre, et ce pour plusieurs programmes qui sont à l émetteur et récepteur.

Programme émetteur:

```
10 CLEAR 1000
20 GOSUB 50000 : REM INITIALISATION
30 BL=12345     : REM QUELQUES VALEURS DE DEMONSTRATION
40 FOR I=0 TO 9
50 B(I)=I+1
60 NEXT I
70 GOSUB 50100 : PASSAGE DES PARAMETRES
80 RUN"1:RECEPT" : REM POUR SYSTEME DISQUE
90 END
50000 A$="":FOR A%=1  TO 3:A$=A$+MKI$(&HA000):NEXT:AN$="******":POKE
VARPTR(AN$)+1,&H C2:POKE VARPTR(AN$)+2,&HF6:LSETAN$=A$:A$="":RETURN
50100   AN$="":POKE   VARPTR(AN$),104:POKE   VARPTR(AN$)+1,&H74:POKE
VARPTR(AN$)+2,&H F6:A$=STRING$(104,0):LSETA$=AN$:RETURN
```

Programme récepteur:

```
10 GOSUB 50200 : REM RECUPERATION DES VARIABLES
20 A$=""       : REMISE A VIDE DE A$
30 PRINT BL    : REM DEMONSTRATION
40 FOR J=1 TO 9
50 PRINT B(I)
60 NEXT J
70 END
50200 A$="":FOR  A%=0 TO 2:POKEVARPTR(A$)+A%,PEEK(&HA000+A%+3):NEXT:
AN$="":POKE VARPTR(AN$),104:POKE VARPTR(AN$)+1,&H74:POKE VARPTR(AN$)
+2,&H F6:LSETAN$=A$:RETURN
```



## 7.15 Quelques exemples de DEF FN.

Pour terminer ce chapitre et par la même occasion, ce
voici quelques exemples de fonctions définies par l'utilisate

A) Calcul du jour courant dans l'année.

```
DEFFNJC%(J%,M%,A%)=(M%-1)*28+VAL(MID$("000303060811131619212424
(M%-1)*2+1,2))-((M%>2)AND((A%ANDNOT-4)=0))+J%
```

Utilisation : pour déterminer le numéro du jour dans
correspondant au 15 mai 1984, écrire :

```
 PRINT FNJC%(15,5,1984)     ->    réponse : 136
```

B) Calcul de la date interne.

```
DEFFNDI!(J%,M%,A%)=A%*365+INT((A%-1)/4)+(M%-1)*28+VAL(MID$("00
60811131619212426",(M%-1)*2+1,2))-((M%>2)AND((A%ANDNOT-4)=0))+
```

Utilisation : identique à A, mais le nombre fourni est un
précision. Cette fonction est valable pour toutes les dates
ses entre 1901 et 2099.

C) Calcul du jour de la semaine.

Cette fonction utilise la précédente, on calcule d'abo
date interne, puis la fonction utilise le résultat pour déte
le jour de la semaine.

```
DEFFNJS$(N!)=MID$("VENDREDISAMEDI    DIMANCHELUNDI     MARDI
REDIJEUDI   ",(N! MOD 7)*9+1,9)
```

Utilisation : 1° calculer la date interne puis la convertir.

```
A!=FNDI!(15,5,1984)
PRINT FNJS$(A!) -> résultat : MARDI
```

# 8 ANNEXES

## 8.1 Caractéristiques générales du BASIC.

Données :

```
Entier            : -32768 à 32767 inclus.
Simple précision  : -9.99999 E-64 à 9.9999999999999 E+62
Double précision  : -9.9999999999999 E-64 à 9.9999999999999 E+62
Chaîne            : 0 à 255 caractères.
```

Numéro de ligne : 0 à 65529 inclus.

Longueur d'une ligne de programme : 255 caractères maximum.

Occupation de la mémoire :

Ligne de programme : minimum 5 octets. 2 octets pour le numéro de ligne, 2 octets pour le pointeur vers la ligne suivante, 1 octet pour marquer la fin de ligne (00). De plus chaque mot clé occupe 1 ou 2 octets, tous les autres caractères occupent 1 octet.

Une variable entière : 5 octets (2 pour la valeur, 2 pour le nom de la variable et 1 pour le type).

Une variable simple précision : 7 octets (4 pour la valeur, 2 pour le nom de la variable et 1 pour le type).

Une variable double précision : 11 octets (8 pour la valeur, 2 pour le nom de la variable et 1 pour le type).

Une variable de chaîne : 6 octets minimum (1 pour le type, 2 pour le nom, 2 pour l'adresse et 1 pour la longueur) + un octet par caractère.

Une variable tableau : 12 octets minimum (2 pour le nom, 1 pour le type, 2 pour la taille, 1 par nombre de dimension, 2 pour chaque dimension et 4 , 6 , 8 ou 16 ( selon le type de la variable ) pour chaque élément du tableau.

Une boucle FOR NEXT : 16 octets

Un GOSUB actif : 5 octets

Un niveau de parenthèses : 16 octets



## 8.2 ANNEXE A Adresse et fonctions des PORTS d'entrée/sortie.

LES PORTS D'ENTREES - SORTIES DU MSX

| ADRESSE | R/W | DESCRIPTION SOMMAIRE | PERI |
|---|---|---|---|
| 80H (128) | R/W | Ecriture/lecture des données sur 8251 | RS23 |
| 81H (129) | R | Lecture du status commande du 8251 | RS23 |
| 82H (130) | R | Lecture des interrrupteurs vitesse | RS23 |
| 83H (131) | R | Lecture des interrupteurs mode | RS23 |
| 83H (131) | W | Ecriture du bit de masque interruption | RS23 |
| 84H (132) | R/W | Compteur 0 du timer 8253 | RS23 |
| 85H (133) | R/W | Compteur 1 du timer 8253 | RS23 |
| 86H (134) | R/W | Compteur 2 du timer 8253 | RS23 |
| 87H (135) | W | Ecriture du mot de commande du 8253 | RS23 |
| 90H (144) | R | Lecture du signal BUSY de l'imprimante | IMPR |
| 90H (144) | E | Ecriture du signal STROBE | IMPR |
| 91H (145) | E | Ecriture du caractère sur l'imprimante | IMPR |
| 98H (152) | R/W | Ecriture/lecture registre VDP | VDP |
| 99H (153) | R/W | Ecriture:commande registre VDP | VDP |
| A0H (160) | W | Ecriture commande PSG | PSG |
| A1H (161) | W | Ecriture registre PSG | PSG |
| A2H (162) | R | Lecture PORT 14 ou 15 PSG | PSG |
| A8H (168) | R/W | Ecriture/lecture du PORT A du PPI | PPI |
| A9H (169) | R/W | Ecriture/lecture du PORT B du PPI | PPI |
| AAH (170) | R/W | Ecriture/lecture du PORT C du PPI | PPI |
| ABH (171) | R/W | Ecriture:lecture du COMMAND-STATUS | PPI |
| B0H - BBH | R/W | Ecriture/lecture du crayon optique SANYO | CROP |
| D0H - D8H | R/W | PORTS réservés au contrôleur de FLOPPYS | FLOPP |
| F7H (247) | W | Ecriture du mot de commande couleur | |

## 8.3 ANNEXE B Table des registres du VDP.

| REG. | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | M3 | EV |
| 1 | 1 | BLANC | IE | M1 | M2 | 0 | SIZE | F. AGR |
| 2 | 0 | 0 | 0 | 0 | ADRESSE DE LA TNP | | | |
| 3 | ADRESSE DE LA TC (TABLE DES COULEURS) | | | | | | | |
| 4 | 0 | 0 | 0 | 0 | 0 | ADRESSE DE LA TGP | | |
| 5 | 0 | ADRESSE DE LA TAS (TABLE ALLOCATION SPRITES) | | | | | | |
| 6 | 0 | 0 | 0 | 0 | 0 | ADRESSE DE LA TGS | | |
| 7 | COULEUR DU TEXTE | | | | COULEUR DU FOND | | | |
| ETAT | F | 5S | COL | No DU 5° SPRITE SUR HORIZONTALE | | | | |

ADRESSE STANDARD DES TABLES.

| MODE | TNP | TGP | TC | TAS | TGS |
|---|---|---|---|---|---|
| TEXTE | 0000H | 0800H | ---- | ---- | ---- |
| GRAPHIQUE 1&2 | 1800H | 0000H | 2000H | 1B00H | 3800H |
| MULTICOLORE | 0800H | 0000H | ---- | 1B00H | 3800H |



## 8.4 ANNEXE C Contenu des registres du PSG.

| REG | UTILISATION | B7 | B6 | B5 | B4 | B3 | B2 | B1 |
|---|---|---|---|---|---|---|---|---|
| R0 | F. CANAL A | 8 BITS DE REGLAGE FIN DE LA FREQUENCE CA | | | | | | |
| R1 | F. CANAL A | ////////////////////// | | | | REGL. GROSSIER CA | | |
| R2 | F. CANAL B | 8 BITS DE REGLAGE FIN DE LA FREQUENCE CA | | | | | | |
| R3 | F. CANAL B | ////////////////////// | | | | REGL. GROSSIER CA | | |
| R4 | F. CANAL C | 8 BITS DE REGLAGE FIN DE LA FREQUENCE CA | | | | | | |
| R5 | F. CANAL C | ////////////////////// | | | | REGL. GROSSIER CA | | |
| R6 | PERIODE BRUIT | //////////////// | | | 5 BITS DE CONTROLE PER | | | |
| R7 | SELECTION | IN OUT | | B R U I T | | | FREQUEN | |
| R8 | AMPLITUDE A | //////////////// | | | M | L3 | L2 | L1 |
| R9 | AMPLITUDE B | //////////////// | | | M | L3 | L2 | L1 |
| R10 | AMPLITUDE C | //////////////// | | | M | L3 | L2 | L1 |
| R11 | PERIODE ENVP. | 8 BITS REGLAGE FIN DE LA PERIODE D'ENVEL | | | | | | |
| R12 | PERIODE ENVP. | 8 BITS REGLAGE GROSSIER PERIODE D'ENVEL | | | | | | |
| R13 | FORME ENVP. | ////////////////////// | | | | CONT | ATT | ALT |
| R14 | PORT E/S A | 8 BITS PORT PARALLELE A (VOIR ANNEXE | | | | | | |
| R15 | PORT E/S B | 8 BITS PORT PARALLELE B (VOIR ANNEXE | | | | | | |

## 8.5 ANNEXE D Utilisation des PORTS du PSG.

| UTILISATION DU PORT A DU PSG AY3-8910 | | |
|---|---|---|
| BIT | PERIPHERIQUE | COMMENTAIRES |
| B0 | JOYSTICK | Vers le HAUT |
| B1 | JOYSTICK | Vers le BAS |
| B2 | JOYSTICK | Vers la GAUCHE |
| B3 | JOYSTICK | Vers la DROITE |
| B4 | JOYSTICK | Bouton de tir |
| B5 | JOYSTICK | Trigger analogique |
| B6 | inutilisé | Version japonaise seulement |
| B7 | CASREAD | Bit de lecture de la cassette |

| UTILISATION DU PORT B DU PSG AY3-8910 | | |
|---|---|---|
| BIT | PERIPHERIQUE | COMMENTAIRES |
| B0-B5 | ANALOGIQUE | Manettes analogiques. |
| B6 | JOYSTICK 1/2 | Selection du joystick 1 ou 2 |
| B7 | inutilisé | Version japonaise uniquement |

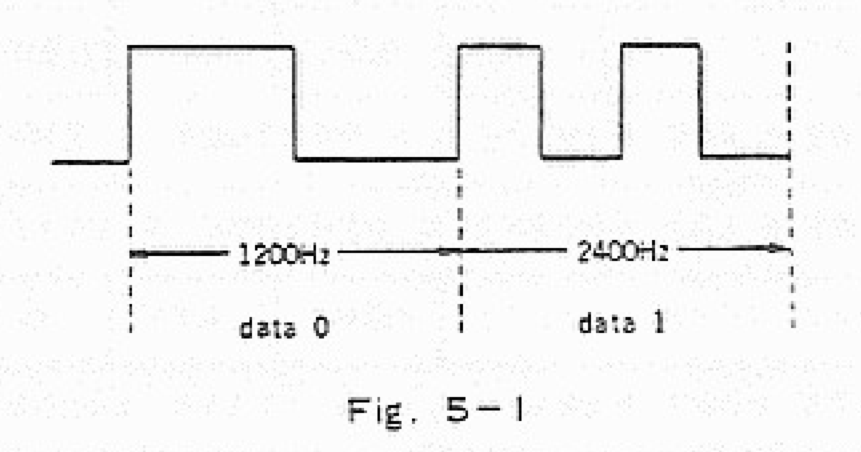
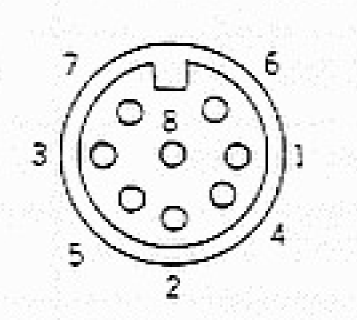


Fig. 5-1

CASSETTE CONNECTOR



(HOSIDEN TCS4460)

And the data format of I byte is as follows.

| start bit | bit 0 | bit 1 | bit 2 | bit 3 | bit 4 | bit 5 | bit 6 | bit 7 | stop bit | stop bit | stop bit |
|---|---|---|---|---|---|---|---|---|---|---|---|

| PIN NO. | SIGNAL | |
|---|---|---|
| 1 | GND | |
| 2 | GND | |
| 3 | GND | |
| 4 | CMTOUT | Red |
| 5 | CMTIN | White |
| 6 | REM+ | Black |
| 7 | REM− | |
| 8 | GND | |

SIGNAUX DU LECTEUR DE CASSETTE



## 8.6 ANNEXE E Programmation des fréquences du PSG.

Table des fréquences pour l'AY3-8910

| Note | Fréq. | R0 | R1 | VL |
|---|---|---|---|---|
| DO | 130,79 | 174 | 6 | 1710 |
| DO# | 138,50 | 79 | 6 | 1615 |
| RE | 146,78 | 244 | 5 | 1524 |
| RE# | 155,44 | 159 | 5 | 1439 |
| MI | 164,80 | 77 | 5 | 1357 |
| FA | 174,61 | 1 | 5 | 1281 |
| FA# | 184,91 | 185 | 4 | 1209 |
| SOL | 195,93 | 117 | 4 | 1141 |
| SOL# | 207,48 | 54 | 4 | 1078 |
| LA | 220 | 248 | 3 | 1016 |
| LA# | 232,98 | 192 | 3 | 960 |
| SI | 246,94 | 138 | 3 | 906 |
| DO | 261,58 | 87 | 3 | 855 |

Remarque : la table est donnée pour les valeurs de DO3 pour la valeurs supérieures ou inférieures, il suffit de div de multiplier la valeur de VL par 2 et de calculer R0 et R1 formule suivante :

R0 = VL MOD 256
R1 = VL \ 256

Exemples :

Produire un SOL2 , la valeur de VL pour SOL3 est de 1141 SOL? elle sera de 2282

=> R0 = 2282 MOD 256 = 234
=> R1 = 2282 \ 256 = 8

Produire un SOL4 , la valeur de VL sera 570.

=> R0 = 570 MOD 256 = 58
=> R1 = 570 \ 256 = 2

## 8.7 ANNEXE F Utilisation des PORTS du PPI.

| UTILISATION DU PORT A DU PPI 8255A | | |
|---|---|---|
| BIT | PERIPHERIQUE | C O M M E N T A I R E S |
| B0 | BANK 0 | Sélection des SLOTS |
| B1 | BANK 0 | " " " |
| B2 | BANK 1 | " " " |
| B3 | BANK 1 | " " " |
| B4 | BANK 2 | " " " |
| B5 | BANK 2 | " " " |
| B6 | BANK 3 | " " " |
| B7 | BANK 3 | " " " |
| UTILISATION DU PORT B DU PPI 8255A | | |
| BIT | PERIPHERIQUE | C O M M E N T A I R E S |
| B0-B7 | CLAVIER | Lecture de la matrice clavier |
| UTILISATION DU PORT C DU PPI 8255A | | |
| B0-B3 | CLAVIER | Ecriture de la ligne clavier à scruter (0 à 8) |
| B4 | CASON | Démarrage du moteur cassette |
| B5 | CASWRITE | Bit d'écriture sur la cassette |
| B6 | CAPS LAMP | Lampe CAPS LOCK 0 = allumee |
| B7 | SOUND | Génération sonore sur 1 bit |

μPD 8255A

| Pin | | Pin | |
|---|---|---|---|
| $PA_3$ | 1 | 40 | $PA_4$ |
| $PA_2$ | 2 | 39 | $PA_5$ |
| $PA_1$ | 3 | 38 | $PA_6$ |
| $PA_0$ | 4 | 37 | $PA_7$ |
| $\overline{RD}$ | 5 | 36 | $\overline{WR}$ |
| $\overline{CS}$ | 6 | 35 | RESET |
| GND | 7 | 34 | $D_0$ |
| $A_1$ | 8 | 33 | $D_1$ |
| $A_0$ | 9 | 32 | $D_2$ |
| $PC_7$ | 10 | 31 | $D_3$ |
| $PC_6$ | 11 | 30 | $D_4$ |
| $PC_5$ | 12 | 29 | $D_5$ |
| $PC_4$ | 13 | 28 | $D_6$ |
| $PC_0$ | 14 | 27 | $D_7$ |
| $PC_1$ | 15 | 26 | Vcc |
| $PC_2$ | 16 | 25 | $PB_7$ |
| $PC_3$ | 17 | 24 | $PB_6$ |
| $PB_0$ | 18 | 23 | $PB_5$ |
| $PB_1$ | 19 | 22 | $PB_4$ |
| $PB_2$ | 20 | 21 | $PB_3$ |

PIN NAMES

| | |
|---|---|
| $D_7$-$D_0$ | Data Bus (Bi-Directional) |
| RESET | Reset Input |
| $\overline{CS}$ | Chip Select |
| $\overline{RD}$ | Read Input |
| $\overline{WR}$ | Write Input |
| $A_0$, $A_1$ | Port Address |
| $PA_7$-$PA_0$ | Port A (Bit) |
| $PB_7$-$PB_0$ | Port B (Bit) |
| $PC_7$-$PC_0$ | Port C (Bit) |
| Vcc | +5 Volts |
| GND | 0 Volts |



## 8.8 ANNEXE G Codes de représentation des mots clés.

### CODES DE REPRESENTATION DES MOTS CLES TRIES PAR CODE.

| DEC | HEX | MOT CLE | DEC | HEX | MOT CLE |
|---|---|---|---|---|---|
| 129 | 81 | END | 130 | 82 | FOR |
| 131 | 83 | NEXT | 132 | 84 | DATA |
| 133 | 85 | INPUT | 134 | 86 | DIM |
| 135 | 87 | READ | 136 | 88 | LET |
| 137 | 89 | GOTO | 138 | 8A | RUN |
| 139 | 8B | IF | 140 | 8C | RESTORE |
| 141 | 8D | GOSUB | 142 | 8E | RETURN |
| 143 | 8F | REM | 144 | 90 | STOP |
| 145 | 91 | PRINT | 146 | 92 | CLEAR |
| 147 | 93 | LIST | 148 | 94 | NEW |
| 149 | 95 | ON | 150 | 96 | WAIT |
| 151 | 97 | DEF | 152 | 98 | POKE |
| 153 | 99 | CONT | 154 | 9A | CSAVE |
| 155 | 9B | CLOAD | 156 | 9C | OUT |
| 157 | 9D | LPRINT | 158 | 9E | LLIST |
| 159 | 9F | CLS,' | 160 | A0 | WIDTH |
| 161 | A1 | ELSE | 162 | A2 | TRON |
| 163 | A3 | TROFF | 164 | A4 | SWAP |
| 165 | A5 | ERASE | 166 | A6 | ERROR |
| 167 | A7 | RESUME | 168 | A8 | DELETE |
| 169 | A9 | AUTO | 170 | AA | RENUM |
| 171 | AB | DEFSTR | 172 | AC | DEFINT |
| 173 | AD | DEFSNG | 174 | AE | DEFDBL |
| 175 | AF | LINE | 176 | B0 | OPEN |
| 177 | B1 | FIELD | 178 | B2 | GET |
| 179 | B3 | PUT | 180 | B4 | CLOSE |
| 181 | B5 | LOAD | 182 | B6 | MERGE |
| 183 | B7 | FILES | 184 | B8 | LSET |
| 185 | B9 | RSET | 186 | BA | SAVE |
| 187 | BB | LFILES | 188 | BC | CIRCLE |
| 189 | BD | COLOR | 190 | BE | DRAW |
| 191 | BF | PAINT | 192 | C0 | BEEP |
| 193 | C1 | PLAY | 194 | C2 | PSET |
| 195 | C3 | PRESET | 196 | C4 | SOUND |
| 197 | C5 | SCREEN | 198 | C6 | VPOKE |

## CODES DE REPRESENTATION DES MOTS CLES (SUITE)

| DEC | HEX | MOT CLE | DEC | HEX | MOT CLE |
|---|---|---|---|---|---|
| 199 | C7 | SPRITE | 200 | C8 | VDP |
| 201 | C9 | BASE | 202 | CA | CALL |
| 203 | CB | TIME | 204 | CC | KEY |
| 205 | CD | MAX | 206 | CE | MOTOR |
| 207 | CF | BLOAD | 208 | D0 | BSAVE |
| 209 | D1 | DSKO$ | 210 | D2 | SET |
| 211 | D3 | NAME | 212 | D4 | KILL |
| 213 | D5 | IPL | 214 | D6 | COPY |
| 215 | D7 | CMD | 216 | D8 | LOCATE |
| 217 | D9 | TO | 218 | DA | THEN |
| 219 | DB | TAB( | 219 | DC | STEP |
| 221 | DD | USR | 221 | DE | FN |
| 223 | DF | SPC( | 223 | E0 | NOT |
| 225 | E1 | ERL | 226 | E2 | ERR |
| 227 | E3 | STRING$ | 228 | E4 | USING |
| 229 | E5 | INSTR | 230 | E6 | ' (REM) |
| 231 | E7 | VARPTR | 232 | E8 | CSRLIN |
| 233 | E9 | ATTR$ | 234 | EA | DSKI$ |
| 235 | EB | OFF | 236 | EC | INKEY$ |
| 237 | ED | POINT | 238 | EE | > |
| 239 | EF | = | 240 | F0 | < |
| 241 | F1 | + | 242 | F2 | - |
| 243 | F3 | * | 244 | F4 | / |
| 245 | F5 | ^ | 246 | F6 | AND |
| 247 | F7 | OR | 248 | F8 | XOR |
| 249 | F9 | EQV | 250 | FA | IMP |
| 251 | FB | MOD | 252 | FC | \ |



## CODES DES MOTS CLES REPRESENTES EN 2 OCTETS

| DEC | HEX | MOT CLE | DEC | HEX | MOT CLE |
|---|---|---|---|---|---|
| 129 | 81 | LEFT$ | 153 | 99 | SPACE$ |
| 130 | 82 | RIGHT$ | 154 | 9A | OCT$ |
| 131 | 83 | MID$ | 155 | 9B | HEX$ |
| 132 | 84 | SGN | 156 | 9C | LPOS |
| 133 | 85 | INT | 157 | 9D | BIN$ |
| 134 | 86 | ABS | 158 | 9E | CINT |
| 135 | 87 | SQR | 159 | 9F | CSNG |
| 136 | 88 | RND | 160 | A0 | CDBL |
| 137 | 89 | SIN | 161 | A1 | FIX |
| 138 | 8A | LOG | 162 | A2 | STICK |
| 139 | 8B | EXP | 163 | A3 | STRIG |
| 140 | 8C | COS | 164 | A4 | PDL |
| 141 | 8D | TAN | 165 | A5 | PAD |
| 142 | 8E | ATN | 166 | A6 | DSKF |
| 143 | 8F | FRE | 167 | A7 | FPOS |
| 144 | 90 | INP | 168 | A8 | CVI |
| 145 | 91 | POS | 169 | A9 | CVS |
| 146 | 92 | LEN | 170 | AA | CVD |
| 147 | 93 | STR$ | 171 | AB | EOF |
| 148 | 94 | VAL | 172 | AC | LOC |
| 149 | 95 | ASC | 173 | AD | LOF |
| 150 | 96 | CHR$ | 174 | AE | MKI$ |
| 151 | 97 | PEEK | 175 | AF | MKS$ |
| 152 | 98 | VPEEK | 176 | B0 | MKD$ |

RAPPEL : LE PREMIER OCTET VAUT TOUJOURS 255 (FFH).

LISTE DES CODES DES MOTS CLES TRIES PAR MOT CLE.

| MOT CLE | HEX | DEC | MOT CLE | HEX | DEC |
|---|---|---|---|---|---|
| * | F3 | 243 | + | F1 | 241 |
| - | F2 | 242 | / | F4 | 244 |
| < | F0 | 240 | = | EF | 239 |
| > | EE | 238 | ABS | 86* | 134 |
| AND | F6 | 246 | ASC | 95* | 149 |
| ATN | 8E* | 142 | ATTR$ | E9 | 233 |
| AUTO | A9 | 169 | BASE | C9 | 201 |
| BEEP | C0 | 192 | BIN$ | 9D* | 157 |
| BLOAD | CF | 207 | BSAVE | D0 | 208 |
| CALL | CA | 202 | CDBL | A0* | 160 |
| CHR$ | 96* | 150 | CINT | 9E* | 158 |
| CIRCLE | BC | 188 | CLEAR | 92 | 146 |
| CLOAD | 9B | 155 | CLOSE | B4 | 180 |
| CLS | 9F | 159 | CMD | D7 | 215 |
| COLOR | BD | 189 | CONT | 99 | 153 |
| COPY | D6 | 214 | COS | 8C* | 140 |
| CSAVE | 9A | 154 | CSNG | 9F* | 159 |
| CSRLIN | E8 | 232 | CVD | AA* | 170 |
| CVI | A8* | 168 | CVS | A9* | 169 |
| DATA | 84 | 132 | DEF | 97 | 151 |
| DEFDBL | AE | 174 | DEFINT | AC | 172 |
| DEFSNG | AD | 173 | DEFSTR | AB | 171 |
| DELETE | A8 | 168 | DIM | 86 | 134 |
| DRAW | BE | 190 | DSKF | A6* | 166 |
| DSKI$ | EA | 234 | DSKO$ | D1 | 209 |
| ELSE | A1 | 161 | END | 81 | 129 |
| EOF | AB* | 171 | EQV | F9 | 249 |
| ERASE | A5 | 165 | ERL | E1 | 225 |
| ERR | E2 | 226 | ERROR | A6 | 166 |
| EXP | 8B* | 139 | FIELD | B1 | 177 |
| FILES | B7 | 183 | FIX | A1* | 161 |
| FN | DE | 222 | FOR | 82 | 130 |
| FPOS | A7* | 167 | FRE | 8F* | 143 |
| GET | B2 | 178 | GOSUB | 8D | 141 |





| MOT CLE | HEX | DEC | MOT CLE | HEX | DEC |
|---|---|---|---|---|---|
| GOTO | 89 | 137 | HEX$ | 9B* | 155 |
| IF | 8B | 139 | IMP | FA | 250 |
| INKEY$ | EC | 236 | INP | 90* | 144 |
| INPUT | 85 | 133 | INSTR | E5 | 229 |
| INT | 85* | 133 | IPL | D5 | 213 |
| KEY | CC | 204 | KILL | D4 | 212 |
| LEFT$ | 81 | 129 | LEN | 92* | 146 |
| LET | 88 | 136 | LFILES | BB | 187 |
| LINE | AF | 175 | LIST | 93 | 147 |
| LLIST | 9E | 158 | LOAD | B5 | 181 |
| LOC | AC* | 172 | LOCATE | D8 | 216 |
| LOF | AD* | 173 | LOG | 8A* | 138 |
| LPOS | 9C* | 156 | LPRINT | 9D | 157 |
| LSET | B8 | 184 | MAX | CD | 205 |
| MERGE | B6 | 182 | MID$ | 83* | 131 |
| MKD$ | B0* | 176 | MKI$ | AE* | 174 |
| MKS$ | AF* | 175 | MOD | FB | 251 |
| MOTOR | CE | 206 | NAME | D3 | 211 |
| NEW | 94 | 148 | NEXT | 83 | 131 |
| NOT | E0 | 224 | OCT$ | 9A* | 154 |
| OFF | EB | 235 | ON | 95 | 149 |
| OPEN | B0 | 176 | OR | F7 | 247 |
| OUT | 9C | 156 | PAD | A5* | 165 |
| PAINT | BF | 191 | PDL | A4* | 164 |
| PEEK | 97* | 151 | PLAY | C1 | 193 |
| POINT | ED | 237 | POKE | 98 | 152 |
| POS | 91* | 145 | PRESET | C3 | 195 |
| PRINT | 91 | 145 | PSET | C2 | 194 |
| PUT | B3 | 179 | READ | 87 | 135 |
| REM | 8F | 143 | RENUM | AA | 170 |
| RESTORE | 8C | 140 | RESUME | A7 | 167 |
| RETURN | 8E | 142 | RIGHT$ | 82* | 130 |
| RND | 88* | 136 | RSET | B9 | 185 |

| MOT CLE | HEX | DEC | MOT CLE | HEX | DEC |
|---|---|---|---|---|---|
| RUN | 8A | 138 | SAVE | BA | 186 |
| SCREEN | C5 | 197 | SET | D2 | 210 |
| SGN | 84* | 132 | SIN | 89* | 137 |
| SOUND | C4 | 196 | SPACE$ | 99* | 153 |
| SPC | DF | 223 | SPRITE | C7 | 199 |
| SQR | 87* | 135 | STEP | DC | 220 |
| STICK | A2* | 162 | STOP | 90 | 144 |
| STR$ | 93* | 147 | STRIG | A3* | 163 |
| STRING$ | E3 | 227 | SWAP | A4 | 164 |
| TAB | DB | 219 | TAN | 8D* | 141 |
| THEN | DA | 218 | TIME | CB | 203 |
| TO | D9 | 217 | TROFF | A3 | 163 |
| TRON | A2 | 162 | USING | E4 | 228 |
| USR | DD | 221 | VAL | 94* | 148 |
| VARPTR | E7 | 231 | VDP | C8 | 200 |
| VPEEK | 98* | 152 | VPOKE | C6 | 198 |
| WAIT | 96 | 150 | WIDTH | A0 | 160 |
| XOR | F8 | 248 | \ | FC | 252 |
| ^ | F5 | 245 | | | |

REMARQUE : les codes inférieurs dont la valeur hexadécimale est suivie d'un astérisque (*) sont des codes représentés sous la forme de 2 octets dont le premier vaut toujours 255 (OFFH).

Pour plus de facilité, celui-ci n'a pas été représenté dans la table.





| MOT CLE | ADR. | MOT CLE | ADR. | MOT CLE | ADR. |
|---|---|---|---|---|---|
| ABS | 2E82 | ASC | 680B | ATN | 2A14 |
| AUTO | 49B5 | BASE | 7B5A | BEEP | 00C0 |
| BIN$ | 65FF | BLOAD | 6EC6 | BSAVE | 6E92 |
| CALL | 55A8 | CDBL | 303A | CHR$ | 681B |
| CINT | 2FBA | CIRCLE | 5B11 | CLEAR | 64AF |
| CLOAD | 703F | CLOSE | 6C14 | CLS | 00C3 |
| CMD | 7C34 | COLOR | 798D | CONT | 6424 |
| COPY | 7C2F | COS | 2993 | CSAVE | 6FB7 |
| CSNG | 2FB2 | CVD | 7C70 | CVI | 5010 |
| CVS | 7C6B | DATA | 485B | DEF | 501D |
| DEFDBL | 4721 | DEFINT | 471B | DEFSNG | 471E |
| DEFSTR | 4718 | DELETE | 53E2 | DIM | 5F9F |
| DRAW | 5D6E | DSKF | 7639 | DSKO$ | 7C16 |
| ELSE | 4850 | END | 63EA | EOF | 6025 |
| ERASE | 6477 | ERROR | 49AA | EXP | 2B4A |
| FIELD | 7C52 | FILES | 6C2F | FIX | 30BE |
| FOR | 2445 | FPOS | 6039 | FRE | 69F2 |
| GET | 775B | GOSUB | 47B2 | GOTO | 47E8 |
| HEX$ | 65FA | IF | 49E5 | INP | 4001 |
| INPUT | 4B6C | INT | 30CF | IPL | 7C2A |
| KEY | 786C | KILL | 7C25 | LEFT$ | 6861 |
| LEN | 67FF | LET | 4880 | LFILES | 6C2A |
| LINE | 4B0E | LIST | 522E | LLIST | 5229 |
| LOAD | 6B50 | LOC | 6D03 | LOCATE | 7766 |
| LOF | 6014 | LOG | 2A72 | LPOS | 4FC7 |
| LPRINT | 4A10 | LSET | 7C48 | MAX | 7E4B |
| MERGE | 6B5E | MID$ | 689A | MKD$ | 7C61 |

## TABLE DES POINTS D'ENTREE DES MOTS CLES

| MOT CLE | ADR. | MOT CLE | ADR. | MOT CLE | ADR. |
|---|---|---|---|---|---|
| MKI$ | 7C57 | MKS$ | 7C5C | MOTOR | 73B7 |
| NAME | 7C20 | NEW | 6286 | NEXT | 6527 |
| OCT$ | 65F5 | ON | 48E4 | OPEN | 6AB7 |
| OUT | 4016 | PAD | 7969 | PAINT | 59C5 |
| PDL | 795A | PEEK | 541C | PLAY | 73E5 |
| POKE | 5423 | POS | 4FCC | PRESET | 57E5 |
| PRINT | 4A24 | PSET | 57EA | PUT | 7758 |
| READ | 4B9F | REM | 485D | RENUM | 5468 |
| RESTORE | 63C9 | RESUME | 495D | RETURN | 4821 |
| RIGHT$ | 6891 | RND | 2BDF | RSET | 7C4D |
| RUN | 479E | SAVE | 6BAB | SCREEN | 79CC |
| SET | 7C1B | SGN | 2E97 | SIN | 29AC |
| SOUND | 73CA | SPACE$ | 6848 | SPRITE | 7A48 |
| SQR | 2AFF | STICK | 794C | SWAP | 643E |
| TAN | 29FB | TIME | 7911 | TROFF | 6439 |
| TRON | 6438 | VAL | 68BB | VDP | 7B37 |
| VPEEK | 7B5F | VPOKE | 7B2E | WAIT | 401C |
| WIDTH | 51C9 | | | | |

Remarque : les mots clés qui ne figurent pas dans cette liste sont les mots clés qui ne peuvent pas se produire seuls (TO , THEN ...), les fonctions logiques (AND, OR, ...) ou les fonctions (VARPTR, USR, POINT, ...).



## 8.9 ANNEXE H Désassembleur.

### PROGRAMME : DESASSEMBLEUR EN BASIC

Il permet de désassembler la ROM ou le stack en haut de mémoi

A la première question : ADRESSE DE DEPART , vous repondez l' absolue à laquelle le désassemblage doit commencer ( 10 exemple ), cette adresse est en décimal par défaut, si vous entrer une adresse en hexadécimal, faites-la précéder par exemple : &HF5D3 ).

La seconde question concerne l'adresse de fin de désassembla même remarque que ci-dessus s'impose.

La troisième question concerne l'utilisation d'une impriman vous répondez O ou OUI à la question, le listing sera f imprimante sinon le listing sera fait sur votre écran.

La quatrième question est différente suivant le choix fai troisième question. Si vous avez choisi l'option imprimant vous demande le nombre de lignes par page (il dépend de votre ou de votre goût). Si vous avez choisi l'option écran o demande si vous voulez un arrêt en bas de page (il f grandement la lecture) , vous répondez par OUI ou par NON.

Si le listing a lieu sur écran, une cinquième question est elle concerne la visualisation des caractères graphiques. savoir que le listing comporte 5 parties, la première l'adresse absolue en hexadécimal, la seconde affiche le code en hexadécimal, la troisième affiche le code objet en ASC quatrième affiche la mnémonique et la dernière les registre déplacement. La troisième partie est concernée par cette q car les caractères qui ne sont pas représentables dans le cod sont transformés en points. Si vous répondez OUI à cette ci question les caractères graphiques sont affichés à l'écran et place des points de remplacement.

```
10 CLS
20 SCREEN 0
30 PRINT"UN MOMENT SVP"
40 CLEAR 2000,&HD000
50 GOSUB 1020
60 INPUT"ADRESSE DE DEPART ";L
70 INPUT"ADRESSE DE FIN ";LF
80 P=0
90 INPUT"IMPRIMANTE (O/N) ";P$
100 INPUT"ARRET EN BAS DE PAGE O/N ";S$
110 S$=LEFT$(S$,1):P$=LEFT$(P$,1)
120 IF P$="O" THEN INPUT"NOMBRE DE LIGNES PAR PAGE ";LP
130 IF P$<>"O" THEN INPUT"AFFICHAGE DES GRAPHIQUES O/N";AG$:AG$=LEFT$(AG$,1)
140 INPUT"tapez enter";RP$
150 IF P$="o"OR P$="O"THEN GOSUB 180 ELSE GOSUB 280
160 IF S$="o"OR S$="O" THEN GOTO 140 ELSE GOTO 150
170 END
180 P=P+1
190 LPRINT CHR$(12)
200 FOR N=1 TO LP
210 GOSUB 340
220 LPRINT L$
230 IF L>LF THEN LPRINT" ":LPRINT"TERMINE":GOTO 2460
240 NEXT N
250 LPRINT" "
260 LPRINT TAB(24);"-";P;"-"
270 RETURN
280 CLS:FOR N=1 TO 22
290 GOSUB 340
300 IF L>LF THEN PRINT:PRINT"TERMINE":GOTO 2460
310 PRINT L$
320 NEXT N
330 RETURN
340 IO=PEEK(L)
350 IF IO=203 THEN 460
360 IF IO=237 THEN 510
370 IF IO=221 THEN 590
380 IF IO=253 THEN 610
390 I1=PEEK(L+1):I2=PEEK(L+2)
400 GOSUB 900
410 RN$="HL":RS$="(HL)"
420 GOSUB 790
430 GOSUB 2220
440 L$=L$+M$
450 RETURN
460 IO=PEEK(L+1)+256
470 GOSUB 900
480 IF M$="???" THEN 430
490 DL=2
500 GOTO 410
```



```
510 IO=PEEK(L+1)
520 IF IO<64 OR (IO>127 AND IO<160) OR IO>191 THEN IO=191
530 IF IO<128 THEN IO=IO+448 ELSE IO=IO+416
540 I1=PEEK(L+2):I2=PEEK(L+3)
550 GOSUB 900
560 IF M$="???" THEN 430
570 DL=DL+1
580 GOTO 410
590 RN$="IX"
600 GOTO 620
610 RN$="IY"
620 C=PEEK(L+2)
630 GOSUB 2400
640 RS$="("+RN$+"+$"+C$+")"
650 IF PEEK(L+1)=203 THEN 730
660 IO=PEEK(L+1):I1=PEEK(L+2):I2=PEEK(L+3)
670 IF IO=54 THEN I1=PEEK(L+3):I2=0
680 GOSUB 900
690 FM=0:FS=0
700 IF M$<>"???" THEN GOSUB 790
710 DL=DL+(FM OR FS)+FS
720 GOTO 430
730 IO=PEEK(L+3)+256
740 GOSUB 900
750 FM=0:FS=0
760 IF M$<>"???" THEN GOSUB 790
770 DL=DL+3*FS
780 GOTO 430
790 FM=0:FS=0:I=5
800 I=I+1:IF I>LEN(M$) THEN RETURN
810 R$=MID$(M$,I,1):IF R$<>"#" AND R$<>"*" THEN 800
820 IF R$="*" THEN 870
830 FM=1
840 M$=LEFT$(M$,I-1)+RN$+RIGHT$(M$,LEN(M$)-I)
850 I=I+LEN(RN$)
860 GOTO 800
870 FS=1
880 M$=LEFT$(M$,I-1)+RS$+RIGHT$(M$,LEN(M$)-I)
890 RETURN
900 IN$=OP$(IO)
910 T=ASC(IN$)-48
920 IF T<1 OR T>9 THEN T=0 ELSE IN$=RIGHT$(IN$,LEN(IN$)-1)
930 FOR I=1 TO LEN (IN$)
940 IF MID$(IN$,I,1)=" " THEN 990
950 NEXT I
960 IO$=IN$+STRING$(5-LEN(IN$)," ")
970 I1$=""
980 GOTO 1010
990 IO$=LEFT$(IN$,I)+STRING$(5-I," ")
1000 I1$=RIGHT$(IN$,LEN(IN$)-I)
```

```
1010 ON T+1 GOTO 1730,1760,1810,1870,1930,1990,2030,2090,2130
1020 DIM OP$(607),FL$(7)
1030 RESTORE
1040 FOR I=0 TO 7 :READ FL$(I):NEXT I
1050 I=0
1060 READ OP$(I)
1070 IF LEFT$(OP$(I),1)<>"1" THEN 1100
1080 FOR J=1 TO 7:OP$(I+J)=OP$(I):NEXT J
1090 I=I+7
1100 I=I+1:IF I<=607 THEN 1060
1110 RETURN
1120 DATA "B","C","D","E","H","L","*","A"
1130 DATA "NOP","3LD BC","LD (BC),A","INC BC","INC B"
1140 DATA "DEC B","2LD B","RLCA","EX AF,AF'","ADD #,BC"
1150 DATA "LD A,(BC)","DEC BC","INC C","DEC C","2LD C"
1160 DATA "RRCA","4DJNZ B","3LD DE","LD (DE),A","INC DE"
1170 DATA "INC D","DEC D","2LD D","RLA","4JR","ADD #,DE"
1180 DATA "LD A,(DE)","DEC DE","INC E","DEC E","2LD E"
1190 DATA "RRA","4JR NZ","3LD #","8LD #","INC #"
1200 DATA "INC H","DEC H","2LD H","DAA","4JR Z","ADD #,#"
1210 DATA "6LD #","DEC #","INC L","DEC L","2LD L","CPL"
1220 DATA "4JR NC","3LD SP","8LD A","INC SP","INC *"
1230 DATA "DEC *","2LD *","SCF","4JR C","ADD #,SP"
1240 DATA "6LD A","DEC SP","INC A","DEC A ","2LD A","CCF"
1250 DATA "1LD B","1LD C","1LD D","1LD E","1LD H","1LD L"
1260 DATA "1LD *","1LD A","1ADD A","1ADC A","1SUB A"
1270 DATA "1SBC A","1AND","1XOR","1OR","1CP"
1280 DATA "RET NZ","POP BC","3JP NZ","3JP","3CALL NZ"
1290 DATA "PUSH BC","2ADD A","RST 0","RET Z","RET"
1300 DATA "3JP Z","9","3CALL Z","3CALL","2ADC A"
1310 DATA "RST $8","RET NC","POP DE","3JP NC"
1320 DATA "7OUT A","3CALL NC","PUSH DE","2SUB A"
1330 DATA "RST $10","RET C","EXX","3JP C","5IN A"
1340 DATA "3CALL C","9","2SBC A","RST $18"
1350 DATA "RET PO","POP #","3JP PO","EX (SP),#"
1360 DATA "3CALL PO","PUSH #","2AND","RST $20"
1370 DATA "RET PE","JP (#)","3JP PE","EX DE,HL"
1380 DATA "3CALL PE","9","2XOR","RST $28"
1390 DATA "RET P","POP AF","3JP P","DI","3CALL P"
1400 DATA "PUSH AF","2OR","RST $30","RET M"
1410 DATA "LD SP,#","3JP M","EI","3CALL M"
1420 DATA "9","2CP","RST $38"
1430 REM CODE CB
1440 DATA "1RLC","1RRC","1RL","1RR"
1450 DATA "1SLA","1SRA","1DPINC","1SRL"
1460 DATA "1BIT 0","1BIT 1","1BIT 2","1BIT 3"
1470 DATA "1BIT 4","1BIT 5","1BIT 6","1BIT 7"
1480 DATA "1RES 0","1RES 1","1RES 2","1RES 3"
1490 DATA "1RES 4","1RES 5","1RES 6","1RES 7"
1500 DATA "1SET 0","1SET 1","1SET 2","1SET 3"
```



```
1510 DATA "1SET 4","1SLT 5","1SET 6","1SET 7"
1520 REM CODE ED40-ED7F
1530 DATA "IN B,(C)","OUT (C),B","SBC HL,BC"
1540 DATA "8LD BC","NEG","RETN","IM 0","LD I,A"
1550 DATA "IN C,(C)","OUT (C),C","ADC HL,BC"
1560 DATA "6LD BC","9","RETI","9","LD R,A"
1570 DATA "IN D,(C)","OUT (C),D","SBC HL,DE"
1580 DATA "8LD DE","9","9","IM 1","LD A,I"
1590 DATA "IN E,(C)","OUT (C),E","ADC HL,DE"
1600 DATA "6LD DE","9","9","IM 2","LD A,R"
1610 DATA "IN H,(C)","OUT (C),H","SBC HL,HL"
1620 DATA "8LD HL","9","9","9","RRD"
1630 DATA "IN L,(C)","OUT (C),L","ADC HL,HL"
1640 DATA "6LD HL","9","9","9","RLD"
1650 DATA "9","9","SBC HL,SP","8LD SP"
1660 DATA "9","9","9","9","IN A,(C)","OUT (C),A"
1670 DATA "ADC HL,SP","6LD SP","9","9","9","9"
1680 REM EDA0-EDBF
1690 DATA "LDI","CPI","INI","OUTI","9","9","9","9"
1700 DATA "LDD","CPD","IND","OUTD","9","9","9","9"
1710 DATA "LDIR","CPIR","INIR","OTIR","9","9","9","9"
1720 DATA "LDDR","CPDR","INDR","OTDR","9","9","9","9"
1730 DL=1
1740 M$=IO$+I1$
1750 RETURN
1760 DL=1
1770 IF LEN(I1$)<>0 THEN I1$=I1$+","
1780 M$=IO$+I1$+FL$(IO AND 7)
1790 IF IO=118 THEN M$="HALT"
1800 RETURN
1810 DL=2
1820 IF LEN(I1$)<>0 THEN I1$=I1$+","
1830 C=I1
1840 GOSUB 2400
1850 M$=IO$+I1$+"0"+C$+"H"
1860 RETURN
1870 DL=3
1880 IF LEN(I1$)<>0 THEN I1$=I1$+","
1890 C=256*I2+I1
1900 GOSUB 2430
1910 M$=IO$+I1$+"0"+C$+"H"
1920 RETURN
1930 DL=2
1940 IF LEN(I1$)<>0 THEN I1$=I1$+","
1950 IF I1<128 THEN C=L+2+I1 ELSE C=L+2+I1-256
1960 GOSUB 2430
1970 M$=IO$+I1$+"0"+C$+"H"
1980 RETURN
1990 DL=2
2000 C=I1
```

```
2010 GOSUB 2400
2020 GOTO 2060
2030 DL=3
2040 C=256*I2+I1
2050 GOSUB 2430
2060 IF LEN(I1$)<>0 THEN I1$=I1$+","
2070 M$=IO$+I1$+"("+"O"+C$+"H)"
2080 RETURN
2090 DL=2
2100 C=I1
2110 GOSUB 2400
2120 GOTO 2160
2130 DL=3
2140 C=256*I2+I1
2150 GOSUB 2430
2160 IF LEN(I1$)<>0 THEN I1$=","+I1$
2170 M$=IO$+"("+"O"+C$+"H)"+I1$
2180 RETURN
2190 DL=1
2200 M$="???"
2210 RETURN
2220 C=L
2230 GOSUB 2430
2240 L$=C$+" "
2250 D$=""
2260 FOR LT=L TO L+DL-1
2270 CT=PEEK(LT)
2280 IF P$="O" OR AG$="N" THEN CT=CT AND 127
2290 IF CT>127 AND CT<160 THEN CT=CT AND 127
2300 IF CT>215 THEN CT=CT AND 127
2310 IF CT<32 OR CT=127 THEN CT=46
2320 L$=L$+CHR$(CT)
2330 C=PEEK(LT)
2340 GOSUB 2400
2350 D$=D$+C$+" "
2360 NEXT LT
2370 L$=L$+STRING$(5-DL," ")+D$+STRING$(3*(5-DL)-2," ")
2380 L=L+DL
2390 RETURN
2400 C$=HEX$(C)
2410 IF LEN(C$)=1 THEN C$="O"+C$
2420 RETURN
2430 C$=HEX$(C)
2440 IF LEN(C$)<4 THEN C$=STRING$(4-LEN(C$),"O")+C$
2450 RETURN
2460 INPUT"TAPEZ ENTER";RP$
2470 CLS
2480 INPUT"ENCORE O/N ";RP$
2490 RP$=LEFT$(RP$,1)
2500 IF RP$="O" OR RP$="o" THEN CLS : GOTO 60
2510 IF RP$<>"N"AND RP$<>"n" THEN GOTO 2470
```



## GLOSSAIRE

BANK : Littéralement BANQUE : morceau de mémoire commutable (voir 1.3 , 4.2.1 , 7.9)

BIOS : BASIC INPUT OUTPUT SYSTEM : système de base pour les entrées et les sorties. (voir 5.12 à 5.15)

MSX : MICROSOFT EXTENDED BASIC : Le standard de votre ordinateur.

PPI : PERIPHERAL PORT INTERFACE : Interface pour PORT périphérique. (voir chapitre 4)

PSG : PROGRAMMABLE SOUND GENERATOR : Générateur de son programmable. (voir chapître 3)

RAM : RANDOM ACCESS MEMORY : Mémoire à lecture et à écriture.

ROM : READ ONLY MEMORY : Mémoire à lecture seule, dans le système MSX elle contient le BIOS et le BASIC.

SLOT : Littéralement FENTE : Dispositif de commutation des BANKS. Au nombre de 4 par BANK (voir 1.3 , 4.2.1 et 7.9)

TAS : TABLE d'ALLOCATION des SPRITES : (voir 2.2.5)

TC : TABLE des COULEURS : (voir 2.2.3)

TGP : TABLE GENERATRICE des PATRONS : (voir 2.2.2)

TGS : TABLE GENERATRICE des SPRITES : (voir 2.2.4)

TNP : TABLE des NOMS de PATRONS : (voir 2.2.1)

VDP : VIDEO DISPLAY PROCESSOR : (voir chapitre 2).

# B I B L I O G R A P H I E .


CHAPITRE II   LE VDP

Texas Instruments 9900 TMS 9918A / TMS 9928A / TMS 9929A.
Vidéo display processors data manual.

CHAPITRE III   LE PSG

Microelectronics data catalog (general instrument)
BELGIQUE Vekano n.v. J.Van Hovestraat 88 - 1950 KRAAINEM.
FRANCE   P.E.P  4 rue Barthélémy - 99120 - MONTROUGE.
SUISSE   Ellyptic  A6 Fellenber Genstrasse 281 - CH-8047 ZURICH
CANADA   Future (Montréal) (514)-694-7710.

CHAPITRE IV   LE PPI

Intel data book microprocessor (MCS 80/85) (8255A)

CHAPITRE V

Microsoft Basic Decoded and others mysteries (IJG).

CHAPITRE VI

Le guide du SPECTRAVIDEO par Lemahieu et Dubois (PSI)
YAMAHA MSX book.
TANDY color computer BASIC.

CHAPITRE VII

La programmation du Z80 par R. Zaks (Sybex).
Le livre du 64 par B. Michel (B.C.M. s.c.).






Comprendre le fonctionnement interne des ordinateurs MSX, transformer le clavier en AZERTY, les messages d'erreurs en français, utiliser tous les modes graphiques, additionner 1000 éléments de vecteur en 1 seconde,...

Découvrez avec ce livre tout ce que vous n'avez jamais rêvé de réaliser sur votre MSX.

# «LE LIVRE DU MSX»

Convient à tous les possesseurs d'ordinateur MSX : BROTHER, CANON, DAEWOO, GENERAL ELECTRIC, GOLDSTART, FUJITSU, HITACHI, ITT, JVC, MITSUBISHI, NEC, PHILIPS, PIONEER, RCA, SANYO, SIEMENS, SONY, SPECTRAVIDEO, TOSHIBA, VICTOR, YAMAHA, YASHICA, YENO,...

Tous les sujets sont abondamment commentés et accompagnés de très nombreux programmes d'exemples en BASIC et en langage machine.

Vous y trouverez un désassembleur, un petit moniteur, un programme d'évaluation de fonctions et un générateur de caractères absolument extraordinaire !

BCM s.c.

24, route de la Sapinière - 4960 Banneux Belgique




ISBN 2-87111-002-6



110 FF.